Master Thesis

# Workflow-management in einem verteilten Simulationssystem

Roland Gude
Fachhochschule Bonn-Rhein-Sieg

DLR

**Fachhochschule Bonn-Rhein-Sieg**

Fachbereich Informatik
Departemet Of Computer Sciences

## Master Thesis

# Workflowmanagement in einem verteilten Simulationssystem

von
Roland Gude

**Erstprüfer:** Prof. Wolfgang Heiden
**Zweitprüfer:** Prof. Manfred Kaul

**Eingereicht am:** 25. Juni 2008

# Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

# Abstract

In der Forschung und Entwicklung finden komplexe Prozesse zur Datenerzeugung, -verarbeitung, -analyse und -visualisierung statt. Oftmals sind an diesen Prozessen Partner aus verschiedenen Einrichtungen beteiligt. Um ihr gemeinsames Ziel zu erreichen, stellen sich die Partner gegenseitig Ressourcen, Daten und Applikationen zur Verfügung. Eine Plattform, die diese Form der Zusammenarbeit erleichtert, wird in der Einrichtung für Simulations- und Softwaretechnik (SISTEC) des Deutschen Zentrums für Luft- und Raumfahrt (DLR) mitentwickelt. Die komplexen Prozesse erfordern aber nicht nur, dass Ressourcen, Daten und Applikationen zur Verfügung stehen, sondern auch, dass diese sinnvoll miteinander verknüpft werden können. Die Verknüpfung mehrerer Applikationen zu einer neuen Applikation, sowie deren Ausführung mit bestimmten Eingabedaten kann durch ein Workflowsystem für die Anwender stark vereinfacht werden. Im Rahmen dieser Master-Thesis wird die erwähnte Plattform um ein solches Workflowsystem erweitert.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

AGWL ..... Abstract Grid Workflow Language
BPEL ..... Buisness Process Execution Language
BPMI ..... Buisness Process Management Initiative
BPMN ..... Buisness Process Modeling Notation
CORBA ..... Common Object Request Broker Architecture
DLR ..... Deutsches Zentrum für Luft- und Raumfahrt
EXSD ..... Eclipse XML Schema Definition
GAP ..... Grid Application Prototype
GAT ..... Grid Application Toolkit
GUI ..... Graphical User Interface
IDL ..... Interface Definition Language
JIT ..... Just-In-Time
JSDL ..... Job Submission Description Language
JXTA ..... juxtapose - eine Peer-To-Peer Architektur von Sun
LEAD ..... Linked Environments for Atmospheric Discovery
MDA ..... Model Driven Architecture
OASIS ..... Organization for the Advancement of Structured Information Standards
OMG ..... Object Management Group
RCE ..... Reconfigurable Computing Environment
RMI ..... Remote Method Invocation
SCAI ..... Fraunhofer Institut Algorithmen und Wissenschaftliches Rechnen
SISTEC ..... Simulations- und Softwaretechnik; Einrichtung des Deutschen Zentrum für Luft- und Raumfahrt
SOAP ..... Früher: Simple Object Access Protocol. Mittlerweile keine Abkürzung mehr.
STP ..... Eclipse Service Tools Platform
UML ..... Unified Modeling Language
WSDL ..... Web Services Description Language
WSL ..... Workflow Specification Language
WSRF ..... Web Service Resource Framework
XML ..... Extensible Markup Language

YAWL ..... Yet Another Workflow Language

# Kapitel 1

# Einleitung

Wissenschaftler verschiedener Domänen analysieren, verarbeiten und visualisieren heutzutage mithilfe von Computer riesige Datenmengen. Die Analyse und Verarbeitung kann hierbei sehr rechenaufwändig sein und spezielle Hardwaresysteme erfordern. Gleiches gilt für die Visualisierung oder das Speichern der Daten. Oft ist es so, dass sich wissenschaftliche Einrichtungen im Rahmen von Projekten zusammenschließen, um die nötigen Ressourcen und das nötige Know-How zur Erreichung eines Ziels zusammenzubekommen. Die boomende Grid-Community[1] ist hierfür ein gutes Beispiel.

Das Arbeiten in solchen Verbünden mit stark verteilten Ressourcen erfordert eine Softwareumgebung, die das gemeinsame Nutzen von Hard- und Software komfortabel erlaubt. In der Einrichtung für Simulations- und Softwaretechnik (SISTEC) des Deutschen Zentrums für Luft- und Raumfahrt (DLR) wird im Rahmen des SESIS-Projektes an einer solchen Softwareumgebung für Schiffbauer und deren Partner gearbeitet.

Diese Master-Thesis beschäftigt sich mit der Konzeption, Implementierung und Integration eines Workflowsystems in diese Softwareumgebung.

Ein Workflowsystem ist eine Software, die es einem Anwender ermöglicht, vorhandene Applikationen miteinander zu verbinden und so neue Applikationen zu schaffen. Ein typischer Anwendungsfall ist hier das Auffinden, Verarbeiten und Visualisieren von Daten. Biologen verfügen über eine über das Web erreichbare Datenbank von Gensequenzen und über mehrere ebenfalls über das Web nutzbare Algorithmen zur Verarbeitung solcher Sequenzen. Angenommen ein Biologe hat eine zu untersuchende Gensequenz und möchte die Ähnlichste in der Datenbank vorhandene Gensequenz neben seiner zu untersuchenden Gensequenz als 3D-Modell visualisieren. Er selbst verfügt aber nur über die Sequenz und ein Visualisierungstool. Um sein Ziel zu erreichen muss er nun die ähnlichste Sequenz in der Datenbank mit Hilfe eines über das Web verfügbaren Algorithmus suchen und die Ergebnissequenz zusammen mit der Ausgangssequenz mit dem Visualisierungstool darstellen. Möchte ein anderer Wissenschaftler das Gleiche tun, muss er ebenfalls diese Schritte durchführen um zum Ziel zu gelangen. Ziel von Workflowsystemen ist es, solche Aufgaben zu automatisieren und die Lösungen weiterhin nutzbar zu machen.

Theoretisch könnte man dieses Problem mit jeder Turingvollständigen Programmiersprache – wie zum Beispiel Java, Python oder C – lösen, allerdings erfordert dies vom Anwender umfangreiche Kenntnisse von den

[1] Als Grid-Community werden die Nutzer und Entwickler des Gridcomputings [Kesselman und Foster (1998)] bezeichnet.

verwendeten Programmiersprachen, Schnittstellen und Protokollen. Eine Anforderung der auch ausgebildete Informatiker oft nicht gewachsen sind. Ein Workflowsystem wird dadurch für den Anwender einfacher zu handhaben, dass es feste Schnittstellen, eine einfache Sprache zum Verbinden von Applikationen – also zum Erstellen von Workflows – und eine Semantik zur Ausführung dieser Workflows definiert. Außerdem kann ein Workflowsystem Möglichkeiten beinhalten, Applikationen aufzufinden oder dafür sorgen, dass Applikationen bei ihrer Ausführung auf der passenden Hardware laufen.

# Kapitel 2

# Anforderungen

Wie in Kapitel 1 beschrieben nutzen Wissenschaftler und Industrieanwender oft komplexe Abläufe um ihre Daten zu visualisieren oder zu verarbeiten. Derartige Arbeit kann durch ein Workflowsystem stark vereinfacht werden. Das RCE-System (siehe Abschnitt 3.1) ist als einheitliche Integrationsplattform für verschiedene Anwendungen und als Plattform zur Zusammenarbeit mehrerer Partner entwickelt worden. Diese Partner können RCE nutzen, um Daten auszutauschen, sich gegenseitig Anwendungen oder Ressourcen zur Verfügung zu stellen und Ähnliches.

Stellen verschiedene Partner Anwendungen bereit und sollen diese Anwendungen im Zusammenspiel miteinander genutzt werden, um zum Beispiel Daten zu analysieren oder zu visualisieren, müssen diese Anwendungen miteinander verknüpft werden. Ein solches Szenario kann von einem Workflowsystem gut unterstützt werden, da es das Verknüpfen der einzelnen Anwendungen ebenso erleichtert wie die Ausführung des resultierenden Workflows. Die typischen Anwendungsfälle für RCE, die das RCE-Workflowsystem betreffen, sind in Abbildung 2.1 als UML-Use-Case-Diagramm dargestellt. Aus den Anwendungsfällen wurden folgende Anforderungen gewonnen:

- Kommunikation mit anderen Anwendungen (Komponenten) muss für Anwendungsentwickler einfach zu realisieren sein.
- Die Wiederverwendbarkeit von Komponenten muss möglichst hoch sein.
- Es muss möglich sein, die einzelnen Komponenten in einem Workflow auf verschiedene Maschinen verteilt auszuführen.
- Es muss möglich sein, die Implementationsdetails einer Komponente vor den Partnern, die diese Komponente nutzen, zu verstecken.
- Workflowkomponenten müssen wie normale RCE-Methoden verwendet werden können.
- Das Erstellen von Workflows aus vorhandenen Komponenten muss über eine graphische Oberfläche möglichst einfach realisierbar sein.
- Die graphische Repräsentationen des Workflows soll es erlauben, einfach auf dessen Funktion zu schließen.

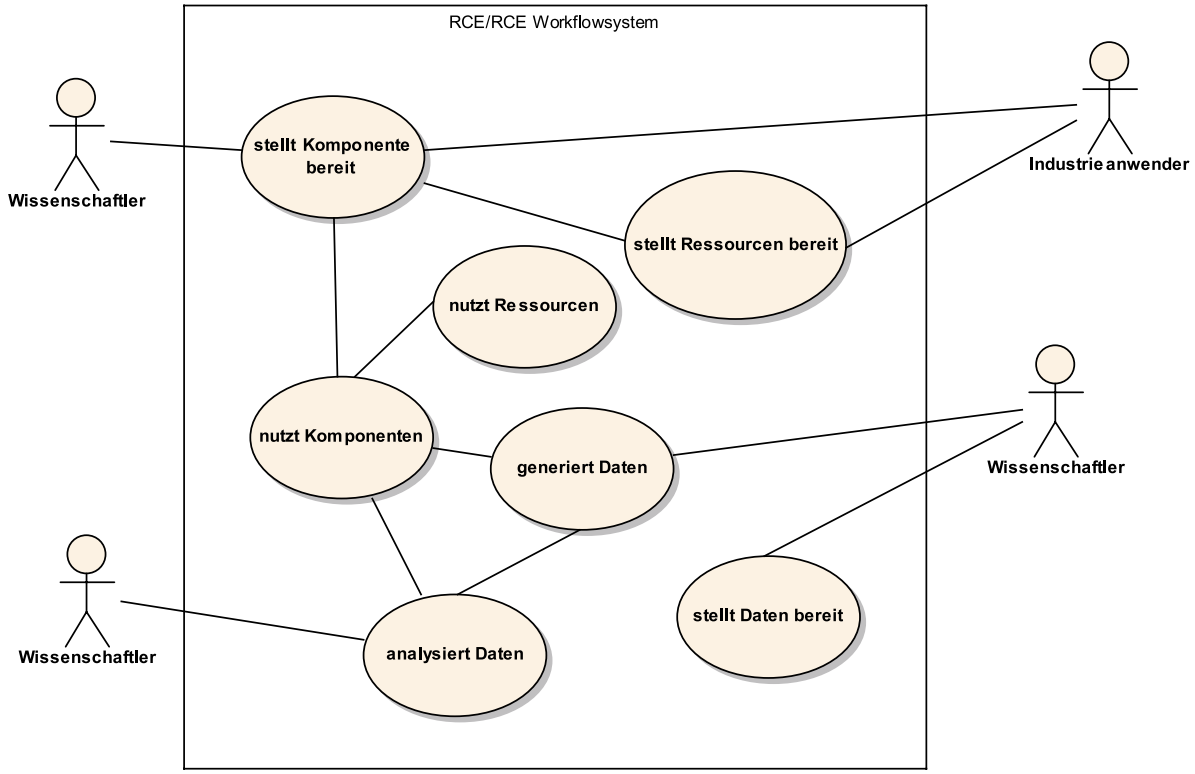



Abbildung 2.1: *UML Use-Case Diagramm für RCE bzw. das RCE-Workflowsystem*

- Das Erstellen und Verändern eines Workflows muss unabhängig von dessen Ausführung sein (Komponenten können miteinander verbunden werden, ohne dass sie tatsächlich gestartet werden).
- Es muss möglich sein, Workflows zu speichern und mehrfach zu starten.
- Workflows müssen aus dem RCE-System heraus bearbeitet werden können.
- Schleifen und Verzweigungen müssen realisierbar sein.
- Kontrollfluss und Datenfluss sollen voneinander getrennt sein. Das heißt, es soll nicht nötig sein sowohl Datenfluss als auch Kontrollfluss gleichzeitig zu modellieren.

Gefordert ist also ein integriertes, verteiltes Workflowsystem mit einer einfach bedienbaren GUI, dessen Ausführung und Repräsentation entkoppelt sind. Ferner müssen die einzelnen Komponenten als Eclipse-Plugins realisiert sein, da das RCE-System auf Eclipse basiert.

# Kapitel 3

# Technologische Basis

Diese Master-Thesis beschäftigt sich mit der Entwicklung eines Workflowsystems für ein verteiltes Simulationsystem. Dieses Kapitel enthält die hierzu nötigen technischen Grundlagen. Hierzu gehören ein Überblick über die verwendete Integrationsplattform und ihre Grundkonzepte sowie Grundlagen darüber, was Workflows sind und welche Arten von Workflows sich unterscheiden lassen.

## 3.1 Das RCE-System

Das Reconfigurable Computing Environment (RCE) ist ein Softwaresystem, welches beim Deutschen Zentrum für Luft- und Raumfahrt (DLR) in der Einrichtung Simulations- und Softwaretechnik (SISTEC) zusammen mit dem Fraunhofer Institut Algorithmen und Wissenschaftliches Rechnen (SCAI) in verschiedenen Forschungsprojekten entwickelt wird. RCE dient als Basisplattform zur Integration verschiedener Anwendungen und kann einfach an die Anforderungen unterschiedlicher Projekte angepasst werden. Mehrere RCE-Instanzen - Installationen der RCE-Software - bilden zusammen das dezentrale und verteilte RCE-System, in dem mehrere Benutzer mit unterschiedlichen Zugriffsrechten arbeiten können. Jede Instanz kann dem Gesamtsystem Funktionalität in Form von RCE-Methoden zur Verfügung stellen.

Zu den wichtigsten Funktionalitäten die RCE bereitstellt, gehören:

- Kommunikation im verteilten RCE-System (zum Beispiel per Java-RMI oder SOAP),
- Rechtemanagement,
- Authentifizierungsmechanismus,
- verteiltes Datenmanagement,
- Starten und Stoppen von RCE-Methoden auf beliebigen RCE-Instanzen (sofern die nötigen Rechte vorhanden sind) und
- umfangreiches Logging.

RCE basiert auf Eclipse. Eclipse [Eclipse-Foundation (2008b)] ist eine von vielen Firmen unterstützte Open-Source-Softwareplattform. Es stellt oft benötigte und wiederverwendbare Softwarekomponenten bereit,

die das Entwickeln eigener Softwareprodukte beschleunigen sollen. Besonders hervorzuheben ist die starke Komponentenorientierung von Eclipse. In Eclipse ist - mit Ausnahme eines kleinen Kerns - jeder Softwareteil ein Plugin und jedes Plugin kann wieder durch Plugins erweitert werden. Hierzu dienen sogenannte Extension-Points und Extensions.

Extension-Points definieren eine Stelle in einem Plugin, die erweitert werden kann. Extensions stellen den Gegenpart dar und können mit Extension-Points verbunden werden, um das Plugin welches den Extension-Point zur Verfügung stellt zu erweitern. Eine Anwendung zum Rendern von Texten, die in einer Markupsprache verfasst sind, könnte zum Beispiel einen Extension-Point besitzen, mit dem die Renderingfunktionen für die unterschiedlichen Markupsprachen[1] verbunden werden. Die eigentliche Anwendung wählt dann für einen Eingabetext die passende Extension von seinem Extension-Point aus und nutzt dessen Funktionalität um die Eingabe in ein der Anwendung verständliches Format zu überführen, welches sie dann darstellt.

Da auf Eclipse basierende Software in Form von solchen Plugins entwickelt wird, kann eine Anwendung relativ einfach um neue Funktionalität erweitert werden. Außerdem ist es durch diesen Ansatz möglich, mit einer auf Eclipse basierenden Anwendung nur jene Komponenten auszuliefern, die auch tatsächlich gebraucht werden. Ein wesentlicher Vorteil des komponentenbasierten Designs ist, dass bei sorgfältigem Einsatz die Wartbarkeit und die Testbarkeit der einzelnen Komponenten deutlich verbessert wird (zum Beispiel durch Entkopplung und übersichtlichere Aufteilung von Funktionen). Eclipse ist in Java geschrieben und basiert auf der OSGi Service Plattform.

Die OSGi Service Platform ist eine auf der Java Virtual Machine aufbauende hardwareunabhängige Plattform zur Entwicklung von komponentenbasierter Software. Eine OSGi-Komponente wird als Bundle bezeichnet und kann Services bereitstellen. Es gibt einen komplexen Lebenszyklus für Bundles. Dieser beinhaltet unter anderem, dass Bundles nur dann tatsächlich gestartet werden, wenn alle benötigten Bundles ebenfalls gestartet werden können. Außerdem erlaubt die OSGi-Service-Plattform das Hinzufügen und Entfernen von Bundles zur Laufzeit.

Die OSGi Service Plattform wird von der *OSGi Alliance* [OSGi-Alliance (2008)], einem Firmenkonsortium dem zum Beispiel IBM, Sun, Nokia und Oracle angehören, spezifiziert.

Es gibt mehrere Implementationen der OSGi Service Platform. Zu ihnen gehören:

- Equinox [Eclipse-Foundation (2008c)]
- Knopflerfish [Makeweave (2008)]
- Oscar (Apache Felix) [Hall (2008); Apache-Software-Foundation (2008)]
- Concierge [ETH-Zürich (2008)]

## 3.2 Workflows

Als Workflow bezeichnet man in der Regel eine Menge von Aufgaben (*Tasks*), die untereinander Abhängigkeiten haben und Daten austauschen können. Der Ursprung heutiger Workflowtechniken liegt bei zwei unabhängigen

[1] wie zum Beispiel reStructuredText [Docutils (2008)] oder Markdown [Gruber (2008)]

Gruppen, die gleichzeitig ähnliche Erfahrungen mit ihrer Arbeit machten und daraus Anforderungen gewannen und Systeme entwickelten, die diesen Anforderungen gerecht werden sollten. Eine Gruppe waren Wissenschaftler, deren Experimente und Forschung aufwändige Datenerzeugung, -verarbeitung, -analyse und -visualisierung beinhalten. Die daraus resultierenden Anforderungen an Rechenleistung musste durch Supercomputer abgedeckt werden. Außerdem standen die verschiedenen Softwarewerkzeuge unter Umständen nur auf unterschiedlichen Systemen zur Verfügung, so dass in einem solchen Experiment Daten zwischen verschiedenen Systemen hin und her kopiert, Programme zur Verarbeitung angestoßen und die Ergebnisse wieder auf andere Systeme transportiert oder archiviert werden mussten. Mit dem Bestreben diese Abläufe zu automatisieren, begann die Entwicklung von Workflowsystemen im wissenschaftlichen Bereich. Die andere Gruppe stammte aus der Geschäftswelt. Dort gab es Anforderungen Geschäftsprozesse zu automatisieren und zu optimieren. Auch hier wurden unterschiedliche Werkzeuge entwickelt um diese Probleme zu lösen [Gannon (2007)].

Im Laufe der Zeit sind zahlreiche Workflowsysteme entstanden (von denen exemplarisch drei Stück in Kapitel 4 vorgestellt werden). Alle Workflowsysteme unterscheiden sich in Mächtigkeit und Einfachheit und eignen sich unterschiedlich gut für bestimmte Problemstellungen. Die Detailunterschiede sind zwar zahlreich, erlauben aber eine grobe Unterteilung in die zwei Gruppen *kontrollflussbasierte Workflowsysteme* und *datenflussbasierte Workflowsysteme* [Shields (2007)]. Diese beiden Gruppen werden in den folgenden Abschnitten kurz vorgestellt. Zudem ist die Workflowsprache ein zentraler Bestandteil eines Workflowsystems, wobei verschiedene Workflowsprachen im letzten Abschnitt des Kapitels vorgestellt werden.

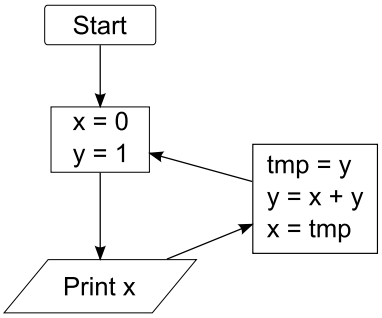
### 3.2.1 Kontrollflussbasierte Workflows

In kontrollflussbasierten Workflows wird der Ablauf des Workflows durch Kontrollstrukturen wie zum Beispiel Sequenzen, Blöcke oder Schleifen definiert. Sie ähneln daher Programmen, die in imperativen Sprachen wie C oder Java geschrieben sind. Üblicherweise gibt es in diesen Workflows *Tasks*, deren Ausführungsreihenfolge explizit vorgegeben wird. Also auf *Task* A folgt *Task* B, auf *Task* B folgen Task C und D, auf Task C folgt Task E und so weiter. Die zur Verfügung stehenden Kontrollstrukturen hängen von der verwendeten Workflowsprache ab.

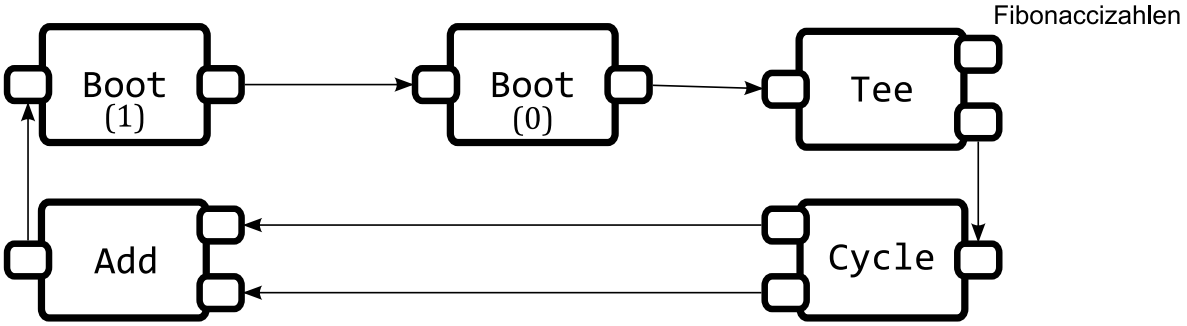
Kontrollflussbasierte Workflows lassen oft einen direkten Zusammenhang zu einem Algorithmus erkennen oder bilden diesen sogar graphisch ab. Abbildung 3.1(a) zeigt ein typisches Flussdiagramm zur Berechnung der Fibonaccizahlen.

### 3.2.2 Datenflussbasierte Workflows

Ein datenflussbasierter Workflow enthält keine explizite Information über die Reihenfolge, in der die einzelnen *Tasks* abgearbeitet werden. Stattdessen wird hier spezifiziert, welche Daten ein *Task* zum Arbeiten benötigt und welche Daten er generiert. Die Workflow-Engine nutzt dann diese Spezifikationen um eine mögliche Ablaufreihenfolge zu ermitteln. Auch in datenflussbasierten Workflows existieren Unterschiede zwischen verschiedenen Systemen. In manchen Systemen bekommen die einzelnen *Tasks* ihre Eingaben in Form von Dateien und erzeugen daraus Ausgabedateien. Diese werden nach Abschluss des erzeugenden *Tasks* an eine Stelle kopiert, an der der nächste *Task* seine Eingabedaten erwartet. Dieser Vorgang wird als *Staging* bezeichnet. Das Work-




(a) Kontrollfluss als typisches Flussdiagramm




(b) Datenfluss als vereinfachte Darstellung der vorgestellten Notation

Abbildung 3.1: *Kontrollfluss im Vergleich zu Datenfluss am Beispiel der Berechnung der Fibonaccizahlen.*

flowsystem des C3-Grids [AWI-Bremerhaven (2008)] arbeitet auf diese Weise (vergleiche Abschnitt 3.2.3). In diesen Systemen können *Tasks* in der Regel erst gestartet werden, wenn alle Eingabedaten komplett erzeugt wurden. In anderen Systemen ist es möglich, dass ein *Task* kleinere Mengen an Ergebnissen an den nächsten *Task* ausliefert, sobald diese zur Verfügung stehen (z.B.: Das *Pipe*-Symbol verbindet zwei Shell Kommandos auf diese Weise. Siehe auch Abschnitt 3.2.3).

Datenflussbasierte Workflows haben meistens nur wenig Ähnlichkeit mit imperativen Algorithmen. Allerdings weisen sie oft eine hohe Ähnlichkeit mit deklarativen oder funktionalen Programmen sowie Schaltplänen in der Elektrotechnik auf [Reekie (1995)]. Abbildung 3.1(b) zeigt ein Datenflussdiagramm zur Berechnung der Fibonaccizahlen[2].

### 3.2.3 Workflowsprachen

Für den Funktionsumfang und die Leistungsfähigkeit eines Workflowsystems ist es unerheblich, ob es kontrollfluss- oder datenflussbasierend arbeitet [Shields (2007)]. Im Gegensatz dazu beeinflusst die verwendete Workflowsprache diese Eigenschaften sehr stark. Es existieren zahlreiche Workflowsprachen. Diese sind in der Regel domänenspezifisch und eng mit bestimmten Problemen verknüpft. Sie stellen unterschiedliche Anforderungen an die verwendbare Hard- und Software. Die Kommunikation zwischen den Komponenten des Workflows ist

[2]Die verwendete Notation ist eine Vereinfachung der in Kapitel 5 vorgestellten Notation für datenflussbasierte Workflows. Die Abbildung wird auch in 5.2 noch einmal verwendet. In diesem Zusammenhang werden auch die einzelnen verwendeten Elemente näher erklärt.

ebenfalls von Sprache zu Sprache anders realisiert [Gannon (2007)]. Im folgenden werden einige Workflowsprachen vorgestellt.

### BPEL

BPEL ist die *Buisness Process Execution Language*. Bei BPEL handelt es sich um eine 2002 von Microsoft, BEA und IBM eingeführte XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren Aktivitäten durch Webservices realisiert werden. Der BPEL Standard nutzt den *WS-**-Protokoll Stack[3]. Die Spezifikation wird bis Version 1.1 als *BPEL4WS* bezeichnet. Die Nachfolgeversion wurde, unter anderem um den Aufbau auf dem WS-*-Stack zu verdeutlichen, in *WS-BPEL* umbenannt. *WS-BPEL 2.0* ist seit 2007 ein OASIS[4]-Standard [Jordan u. a. (2007)]. *WS-BPEL* und *BPEL4WS* sind inkompatibel zueinander [Alves (2005)].

BPEL-Beschreibungen werden von einer BPEL-Engine ausgeführt. Da die einzelnen Aktivitäten durch Webservices implementiert werden und Webservices aufgrund ihres Designs leicht verteilbar sind[5], ist es einfach verteilte Workflows zu definieren. Das Deployment von Webservices wird allerdings durch BPEL nicht spezifiziert.

Die Wiederverwendbarkeit von Workflows soll dadurch erreicht werden, dass BPEL-Workflows selbst wieder Webservices sind und somit als Aktivitäten in anderen Workflows genutzt werden können.

BPEL ist hauptsächlich kontrollflussgesteuert. Allerdings gibt es ein Kontrollflusselement *flow*, in welchem alle Aktivitäten in einer beliebigen Reihenfolge ausgeführt und Abhängigkeiten durch *links* angegeben werden. Dies erlaubt es eingeschränkte datenflussbasierte Workflows mit BPEL zu erstellen. Es ist daher möglich datenflussbasierte und kontrollflussbasierte Workflows in BPEL miteinander zu kombinieren.

BPEL wird als Basis wissenschaftlicher Workflows im LEAD-Projekt (Linked Environments for Atmospheric Discovery) [Slominski (2007); Gannon u. a. (2007)] verwendet.

### BPMN

BPMN ist die *Buisness Process Modeling Notation*. Hierbei handelt es sich um eine rein graphische Sprache. Die BPMN wurde entwickelt um Buisness-Workflows graphisch darzustellen und zu modellieren. Außerdem unterstützt es *Object-Properties*, die es ermöglichen sollen, ausführbaren BPEL-Code zu generieren. Es existiert eine Abbildung von BPMN 1.0 auf BPEL 1.1 (BPEL4WS) [White (2005)].

[3] Als WS-* Protokoll Stack werden die verschiedene Protokolle für Webservices bezeichnet, die jeweils unterschiedliche Anforderungen an Webservices behandeln. Hierzu gehören zum Beispiel *SOAP* als Remote Method Invocation Protokoll, Die *Web Services Description Language (WSDL)* zur Definition von Schnittstellen. Weniger bekannt sind die meisten für den Namen verantwortlichen Protokolle deren Namen mit *WS-* beginnen. Hierzu gehören z.B. *WS-Notification*, *WS-Security*, *WS-Trust*, *WS-Addressing* und viele andere [Booth u. a. (2004); Champion u. a. (2002–2004); Lafon (2002–2008)].

[4] OASIS ist ein Standardisierungsgremium. Die Abkürzung steht für *Organization for the Advancement of Structured Information Standards*. Namenhafte Mitglieder von OASIS sind zum Beispiel IBM, BEA und Sun. Die OASIS ist hauptsächlich im Bereich Webservices tätig. Zu den OASIS-Standards gehören zum Beispiel die dem WS-*-Stack zugehörigen Protokolle WS-Security und WS-Trust. Siehe auch [OASIS (1993–2008)].

[5] Viele der WS-* Protokolle adressieren allgemeine Anforderungen an verteilte Systeme.

BPMN 1.0 wurde von der BPMI (Buisness Process Management Initiative) im Februar 2004 fertiggestellt [White u. a. (2004)]. Seit die BPMI 2005 mit der OMG[6] fusionierte, wird die Spezifikation von der OMG gepflegt, welche 2006 eine *Final Adopted Specification* [OMG (2006)] veröffentlichte. Derzeit wird bei der OMG die Arbeit an BPMN 2.0 vorbereitet [OMG (2007)].

### YAWL

YAWL [YAWL (2008)] ist die Abkürzung für *Yet Another Workflow Language*. Ziel von YAWL ist es, die von der Workflow-Patterns-Initiative spezifizierten Workflow-Patterns [Workflow-Patterns-Initiative (2007)] zu unterstützen. YAWL ist wie BPMN eine graphische Sprache. Sie basiert allerdings auf Petri-Netzen. Petri-Netze werden oft zur Modellierung von Workflows genutzt, da sie geeignet sind Algorithmen abzubilden und mit mathematischen Werkzeugen analysiert werden können. Es existieren so genannte High-Level-Petri-Netze die Turing-Vollständig sind[7] [Hoheisel und Alt (2007); van der Aalst und ter Hofstede (2003)].

Die Entwickler von YAWL sind der Meinung, dass Petri-Netze zwar sehr gut für zustandsbasierte Workflows geeignet sind, geben aber zu bedenken, dass sich einige identifizierte Workflow-Patterns nur schwer auf Petri-Netze abbilden lassen. YAWL baut auf Petri-Netzen auf und kann auch die übrigen Patterns einfach abbilden. YAWL selbst kann auf Petri-Netze abgebildet werden, was allerdings nach Aussage der Autoren sehr schwierig ist [van der Aalst und ter Hofstede (2005, 2003)].

Die YAWL-Autoren haben auch eine Referenzimplementierung eines Workflowsystems fertiggestellt, die YAWL als Workflowsprache nutzt [van der Aalst u. a. (2004, 2003)]. Auf Basis der Erkenntnisse der Autoren aus YAWL wird derzeit mit *new*YAWL [Russel u. a. (2007)] eine neue Workflowsprache entwickelt. Sowohl YAWL als auch *new*YAWL sollen Datenfluss darstellen können, wobei sich die bisher veröffentlichten Paper auf den Kontrollfluss beschränken.

### JSDL und WSL

Die Job Submission Description Language (JSDL) ist eine Sprache um Gridcomputing Jobs zu beschreiben [Anjomshoaa u. a. (2005)]. Zu einer Jobbeschreibung gehören zum Beispiel Ressourcenanforderungen und der Pfad zu einer ausführbaren Datei. Außerdem gehören Aufrufparameter (z.B.: die Kommandozeilenparameter einer ausführbaren Datei) zu der Jobbeschreibung. Die JSDL kann nicht als Workflowsprache bezeichnet werden, da sie lediglich zur Beschreibung von einzelnen Jobs dient. Abhängigkeiten zu anderen Jobs können ebenso wenig spezifiziert werden wie Datenabhängigkeiten oder Kontrollfluss.

Die JSDL ist hier dennoch aufgeführt, da sie im Gridumfeld weit verbreitet ist und Ansätze existieren, die die JSDL zu einer Workflowsprache erweitern. Hierzu gehört die Workflow Specification Language (WSL) [Grimme und Papaspyrou (2006)]. Diese erweitert die JSDL um Dateiabhängigkeiten und Informationen, welche Dateien ein Job erzeugt. Die Dateien müssen dann mittels Staging an den Orten abgelegt werden, wo nachfolgende Jobs sie erwarten. Eine Workflow-Engine kann so aus einer Menge von WSL-Dateien einen Workflow aufbauen.

[6] Die OMG (*Object Management Group*), ist ein Firmenkonsortium dem zum Beispiel Sun, IBM und Apple angehören. Die OMG entwickelt international anerkannte Standards. Zum Beispiel CORBA, IDL, UML und MDA wurden in der OMG entwickelt [OMG (1997–2008)]

[7] Das heißt, man kann mit diesen Netzen alles ausdrücken/berechnen was man mit einem Algorithmus ausdrücken/berechnen kann.

### Martlet

Martlet ist eine Workflowsprache, die wesentlich von funktionalen Programmiersprachen, wie zum Beispiel Haskell, beeinflusst wurde [Goodman (2006)]. Ein Ziel von Martlet ist es, die Schwierigkeiten paralleler und verteilter Programmierung zu verstecken. Hierzu werden, neben den aus der funktionalen Programmierung übernommenen Konstrukten, *lokale* und *verteilte* Datenstrukturen unterschieden. Eine verteilte Datenstruktur kann man dabei als Menge von Referenzen zu lokalen Datenstrukturen auffassen. Eine Funktion in Martlet besteht aus zwei Teilen, dem Basisfall und dem induktiven Fall. Im Basisfall wird angegeben, wie die Daten in einer lokalen Datenstruktur verarbeitet werden sollen. Im induktiven Fall wird angegeben, wie die Ergebnisse der verschiedenen Verarbeitungen zusammengefasst werden sollen. Zur Zeit ist ein JIT-Compiler angedacht, der Martlet-Programme in andere Workflowsprachen übersetzen kann [Goodman (2007)].

Martlet wird derzeit mit Daten des ClimatePrediction.net-Projektes [ClimatePrediction.net (2008)] getestet.

### Shell-Skript

Shell-Skripte wie sie in der Unix-Welt üblich sind, können als Workflowbeschreibungen betrachtet werden. Hierbei bilden einzelne ausführbare Dateien die Komponenten, welche über Standardeingabe und Standardausgabe kommunizieren. Das Pipe-Symbol dient als Verbindung zwischen zwei Komponenten. Zusammen mit den in Shell-Skripten zur Verfügung stehenden Kontrollstrukturen, wie Zählschleifen und bedingten Ausführungen sowie den zahlreichen und qualitativ hochwertigen Komponenten, bilden Shell-Skripte ein mächtiges hybrides Workflowsystem.

Durch verteilte Dateisysteme wie zum Beispiel NFS und Tools wie ssh sind sogar verteilte Workflows möglich. Die einzelnen Komponenten sind allerdings in der Regel in ihrer Funktion sehr limitiert. Komponenten, die komplexe Aufgaben lösen und sich an die Shell-Skript-Konventionen halten, sind eher selten. Daher müssen Workflows normalerweise aus den existierenden Low-Level-Komponenten erstellt werden, so dass das Erstellen von Shell-Skripten einiges an Erfahrung erfordert. Außerdem sind Shell-Skripte sehr stark an eine bestimmte Umgebung gebunden und sehr systemnah. Aus diesen Gründen sind sie in der Regel nicht für die Szenarien geeignet, in denen Workflowsysteme eingesetzt werden.

### AGWL

AGWL ist die Abstract Grid Workflow Language [Fahringer u. a. (2004, 2005, 2007)]. Sie kommt in dem ASKALON-Workflowsystem (siehe Abschnitt 4.2) zum Einsatz. Die Sprache basiert auf XML und verfügt über eine graphische Repräsentation, die auf UML basiert [Qin u. a. (2007)]. Die AGWL wurde mit besonderem Hinblick auf Gridcomputing entwickelt. Insbesondere der dynamischen Verfügbarkeit bzw. Nichtverfügbarkeit von Ressourcen wurde Rechnung getragen. Die Komponenten in einem AGWL-Workflow werden als *Activities* bezeichnet. Die Implementierung dieser Activities ist nicht an eine bestimmte Technologie gebunden. Es wird zwischen *Activity Types* und *Activity Deployments* unterschieden. Activity Types bezeichnet die Semantik und die Funktion einer Activity (ähnlich einer Interfacebeschreibung). Activity Deployments bezeichnen eine tatsächlich ausführbare Activity (zum Beispiel ein Executable). Für AGWL sind lediglich die Activity Types

sichtbar.

Die AGWL verfügt über zahlreiche Kontrollflusselemente (zum Beispiel mehrere Schleifenarten, parallele Schleifen, If-Then-Else-Konstrukte etc.) und bringt mit den so genannten *Input*- und *Outputports* einen Mechanismus zur Modellierung von Datenfluss mit. AGWL ist also eine hybride Workflowsprache.

# Kapitel 4

# Ähnliche Softwareprodukte

Workflowsysteme werden in vielen Projekten verschiedener Domänen oft benötigt. Aus diesem Grund gibt es zahlreiche Projekte, sowohl im wissenschaftlichen als auch im industriellen Bereich, in denen Workflowsysteme erforscht und entwickelt werden. Hierbei gibt es kommerzielle und frei verfügbare Produkte. In diesem Kapitel werden drei Workflowsysteme vorgestellt, die von verschiedenen Forschergruppen seit mehreren Jahren entwickelt werden.

## 4.1 Kepler

*Kepler* [Kepler-Project (2003–2008)] ist ein Workflowsystem für wissenschaftliche Workflows. Ein wesentliches Ziel von Kepler ist es, den Wissenschaftlern die Arbeit mit dem Grid – zum Beispiel durch Abstraktion der Gridmiddleware und konkreter Gridtechnologien – zu vereinfachen. Besonderes Augenmerk wird auf ein intuitives graphisches Benutzerinterface gelegt. Außerdem soll das Workflowsystem domänenunabhängig in allen wissenschaftlichen Disziplinen einsetzbar sein. Hierzu zählen die Kepler-Entwickler sowohl Biologen, Chemiker und Physiker, die komplexe Datenanalysen durchführen wollen ohne mit der zugrunde liegenden Technologie konfrontiert zu werden, als auch Entwickler, die direkt an dem Grid arbeiten und Workflows zum Beispiel als Systemtests im Gridcomputing verwenden [Altintas u. a. (2004); Ludäscher u. a. (2006)].

In Kepler werden Workflows mit einem Fokus auf den Datenfluss modelliert. Zusätzlich stehen dem Anwender Kontrollflusselemente zur Verfügung. In Kepler heißen die Komponenten eines Workflows *Actors*. Ein *Actor* hat eine Menge an *input*- und *outputports*. Diese Ports sind typisiert. Hierbei wird in Kepler zwischen *structural type* und *semantic type* unterschieden. Als *semantic type* wird eine Typisierung mit einer Ontologiesprache[1] bezeichnet, die zum Beispiel aussagt, dass ein Port Temperaturmesswerte akzeptiert. Der *structural type* ist ein Datentyp wie zum Beispiel der in XML-Schema definierte *xsd:String*. Weder *structural type* noch *semantic type* sind an ein konkretes Typsystem beziehungsweise an eine konkrete Ontologiesprache gebunden. Ob ein *inputport* mit einem *outputport* verbunden werden kann, ermittelt Kepler über die konkreten Typsysteme, die verwendet werden, und hängt davon ab, ob – sofern nicht sowieso die gleichen Typen verwendet werden – Adapter existieren, die den einen Typ auf den anderen abbilden können [Bowers und Ludäscher (2005)].

[1] Eine Ontologie spezifiziert ein System von Begriffen. Mit einer Ontologiesprache kann man solche Ontologien formal definieren [Hesse (2002)].

Eine Besonderheit von Kepler ist es, dass die Ausführungssemantik eines Workflows nicht durch die Komponenten oder das System vorgegeben ist, sondern durch so genannte *Directors* realisiert wird [Bowers und Ludäscher (2005)]. Jeder Workflow in Kepler besteht nicht nur aus den Komponenten und den Datenflussverbindungen, sondern verfügt auch über einen solchen *Director*. Dieser beeinflusst, wie sich die Komponenten tatsächlich verhalten und wann Komponenten zum Beispiel lesen oder schreiben können oder ob Lese- und Schreiboperationen blockieren oder nicht. Im Gegensatz zu Kontrollfusselementen wie Schleifen wird hier nicht beeinflusst, wann und wie oft Komponenten gestartet werden, sondern wie sich die Komponenten verhalten und wie sie mit anderen Komponenten kommunizieren. Zu den zur Verfügung stehenden *Directors* gehören:

- Synchronous Dataflow - Komponenten kommunizieren über Datenflussverbindungen und senden/empfangen immer eine feste Anzahl von Datenelementen. Das Senden/Empfangen erfolgt nach einem vorab ermittelten statischen Plan.
- Process Network - Jede Komponente wird in einem eigenen Thread oder Prozess ausgeführt. Datenflussverbindungen repräsentieren unbegrenzte Warteschlangen. Ausgaben können jederzeit geschrieben werden, Lesen ist eine blockierende Operation.

Komponenten in Kepler haben einen recht komplexen Lebenszyklus und müssen eine Reihe von Methoden implementieren, um von einem *Director* korrekt angesteuert werden zu können [Ludäscher u. a. (2006)]. Zu den Methoden, die eine Komponente implementieren muss, gehören:

- *preinitialize* - Wird nur einmal pro Instantiierung einer Komponente aufgerufen, selbst wenn der Workflow mehrfach ausgeführt wird. Diese Methode dient zum Beispiel dazu *input*- und *outpuports* bekanntzugeben.
- *initialize* - Wird bei jeder Ausführung eines Workflows erneut aufgerufen.
- *prefire, fire, postfire* - Jede Iteration in einer Workflowausführung beinhaltet einen Aufruf jeder dieser Methoden. Die *fire*-Methode soll hierbei die eigentlich Applikationslogik enthalten und kann von bestimmten *Directors* auch mehrfach aufgerufen werden.

## Beurteilung

Kepler ist ein sehr flexibles Workflowsystem und lässt dem Workflowentwickler nahezu alle Freiheiten, zur Definition eines Workflows. Allerdings geht diese Flexibilität auf Kosten der Einfachheit. Insbesondere der komplexe Lebenszyklus und die damit verbundene Anzahl der Operationen, die ein Komponentenentwickler zu implementieren hat, erscheint problematisch. Die Semantik des Workflows in die so genannten *Director*-Komponenten auszulagern erhöht zwar möglicherweise, wie von den Entwicklern versprochen, die Wiederverwendbarkeit der Komponenten, erschwert aber auch die Definition und das Interpretieren von Workflows, da die Semantik bei nahezu gleichem Workflowdiagramm (es unterscheidet sich nur die *Director*-Komponente) stark variieren kann. Ob die Implementierung von Komponenten durch dieses Konzept tatsächlich vereinfacht wird, bleibt fraglich, da ein Komponentenentwickler die Semantik zwar nicht selber implementieren muss, aber bei der Implementierung seiner Applikationslogik auch die unterschiedlichen Ausführungssemantiken beachten muss.

Letztendlich muss auch der Workflowentwickler eine nicht zu unterschätzende Kenntnis der mit einem Director verbundenen Semantik mitbringen und er muss in der Lage sein, zu beurteilen, unter welchem *Director* eine Komponente verwendbar ist und unter welchem nicht.

Zur Integration in RCE ist Kepler nicht geeignet, da es sowohl von einem Workflowentwickler als auch von einem Komponentenentwickler zu viel Wissen über Kepler verlangt. Kepler soll zwar die Anforderungen von Wissenschaftlern ohne Informatikhintergrund erfüllen, versucht aber den Spagat, auch denjenigen die die unter dem Workflowsystem liegende Gridmiddleware untersuchen wollen alle Freiheiten zu lassen. Die hierfür notwendige Komplexität kann Kepler nicht ausreichend verbergen.

## 4.2 Askalon

Das *ASKALON Grid application development and computing environment* ist ein Workflowsystem das seine stark auf Gridcomputing ausgerichtet ist [Fahringer u. a. (2007)]. Die Entwickler von Askalon legen besonderen Wert darauf, Gidcomputing für Anwendungsentwickler transparenter zu gestalten als andere Entwicklungsumgebungen. Als problematisch empfinden sie besonders, dass es in anderen Umgebungen zu den üblichen Aufgaben eines Grid-Anwendungsentwicklers gehört explizit Softwarekomponenten und die zu nutzende Hardware auszuwählen und darum bietet Askalon in diesem Bereich Unterstützung. Desweiteren wollen sie eine Entwicklungsumgebung bereitstellen, die dem Anwendungsentwickler keine Vorgaben darüber macht, mit welcher Technologie er Komponenten implementiert und hierfür eine Middleware anbieten, die von den Systemdetails stärker abstrahiert als andere Entwicklungsumgebungen. Außerdem legen sie Wert darauf, dass die graphische Workflowmodellierung auf Standardtechnologien aufbaut. Das Ziel von Askalon ist es, dem Anwendungsentwickler ein *unsichtbares Grid* zur Verfügung zu stellen.

Workflows werden in Askalon in der AGWL (siehe Abschnitt 3.2.3) spezifiziert. Mithilfe von AGWL und ihrer UML-basierten graphischen Repräsentation werden bereits einige der angesprochenen Probleme gelöst. In der AGWL werden zum Beispiel lediglich die abstrakten *Activity Types* anstatt der konkreten *Activity Deployments* verwendet. Ein Anwendungsentwickler muss also nur spezifizieren, welche Aufgabe eine Workflowkomponente erfüllen soll, aber nicht, wie sie implementiert ist oder wo sie deployed wurde. Selbst wenn ein Workflow bestimmte Systemanforderungen hat, muss die Hardware nicht vom Entwickler ausgesucht werden. In diesem Fall reicht es, Constraints für Komponenten und Datenwege zu definieren (z.B.: mindestens 1 Gigabit/s Netzwerkdurchsatz auf einem Datenweg). Askalon nutzt diese Constraints um aus den im Grid zur Verfügung stehenden Ressourcen die passenden auszuwählen.

Die Askalon-Laufzeitumgebung besteht aus einer Reihe von auf WSRF (Web Service Resource Framework) basierenden Services. Diese sind:

- *Resource Manager*: Zuständig für die Reservierung von Ressourcen, das Deployment von Services. Dient (zusammen mit AGWL) als Abstraktion der systemnahen Gridmiddleware [Siddiqui und Fahringer (2005); Fahringer u. a. (2007)]

- *Scheduler*: Bildet die zu startenden Workflowbeschreibungen möglichst effizient auf die Gridressourcen ab. Bietet *Quality Of Service* durch dynamisches Anpassen der Abbildungen, falls sich die Gridinfrastruk-

tur zur Laufzeit ändert. Nutzt *Performance Prediction* und *Resource Manager* [Wieczorek u. a. (2005); Fahringer u. a. (2007)].

- *Execution Engine*: Kontrolliert die Ausführung der vom *Scheduler* erstellten Pläne. Nutzt verschiedene Techniken, um die Zuverlässigkeit und Fehlertoleranz der Ausführung zu erhöhen [Duan u. a. (2005); Fahringer u. a. (2007)].

- *Performance Analysis*: Instrumentiert Workflows automatisch und hilft bei der Erkennung von Bottlenecks [Nadeem u. a. (2007); Fahringer u. a. (2007)].

- *Performance Prediction*: Macht, mit Hilfe der *Performance Analysis* und einer Trainingsphase, Vorhersagen über die Ausführdauer von Aktivitäten [Nadeem u. a. (2006); Fahringer u. a. (2007)].

### Beurteilung

Askalon ist ein sehr komplexes Workflowsystem mit zahlreichen Funktionen und klaren Konzepten. Es verfügt über ausgereifte Scheduler und Performanceanalysewerkzeuge. Allerdings ist es sehr stark auf Gridcomputing zugeschnitten und benötigt zum Starten von Workflows immer Gridressourcen. RCE soll zwar mit Grids zusammenarbeiten, d.h. es soll möglich sein Jobs in einem Grid abzusetzen, aber es soll auch funktionsfähig sein, ohne dass komplexe Gridumgebungen benötigt werden. Obwohl der Funktionsumfang und die Leistungsmerkmale von Askalon sehr beeindruckend sind, kommt es daher für den RCE-Anwendungsfall nicht in Frage. Denkbar ist allerdings, dass Askalon-Workflows als RCE-Komponenten abgebildet werden und somit, falls ein Grid zur Verfügung steht, Gridworkflows als Teilworkflows in RCE-Workflows eingesetzt werden.

## 4.3 Triana

Triana ist ein Workflowsystem, das aus einer graphischen Benutzerschnittstelle und einem Subsystem besteht [Taylor u. a. (2007)]. Das Subsystem ist über das Grid Application Toolkit (GAT) an verschiedene Gridmiddlewarelösungen und über das Grid Application Prototype (GAP) Interface an verschiedene Peer-To-Peer-Technologien angebunden.

Ein Workflow in Triana besteht aus mehreren *Tools* die über *Input-* und *Outputports* verfügen. Diese werden graphisch miteinander verbunden. Als Tool kann jede Software genutzt werden, die über GAT oder GAP erreichbar ist. Dies sind zum Beispiel Webservices oder Jobs in einem Globus-Grid[2]. Es ist auch möglich in einem Workflow sowohl Komponenten zu nutzen, die im Grid laufen, als auch solche, die über einen Webservice realisiert oder über ein Peer-To-Peer-Netz erreichbar sind. Außerdem verfügt Triana über ein System zur Integration von Legacy-Anwendungen.

Trianas Workflowsprache unterscheidet sich von anderen Workflowsprachen dadurch, dass sie keine Kontrollflusselemente zur Verfügung stellt. Schleifen und Verzweigungen müssen über bestimmte Komponenten nachgebaut werden. Triana kann Workflowbeschreibungen von anderen Workflowsystemen mit Import-Plugins importieren.

[2] Globus ist eine Gridmiddleware [Globus-Alliance (2008)]

Triana verwendet GAT um die Ausführung der Komponente zu und das Staging von Dateien zu koordinieren. Es kann also als graphische Benutzeroberfläche für die darunter liegenden GAT-Ausführungsmechanismen und als Brücke von Komponenten, die über GAT erreichbar sind, zu Komponenten, die über andere Technologien erreichbar sind, gesehen werden.

### Beurteilung

Triana kann Komponenten aus sehr unterschiedlichen Systemen integrieren und verzichtet völlig auf Kontrollflusselemente. Die Workflowrepräsentation ist sehr einfach und leicht zugänglich. Aufgrund der vielen möglichen Subsysteme ist der Ablauf eines Workflows aber unter Umständen nicht leicht nachzuvollziehen.

Triana lässt sich nur schwer in RCE integrieren, da viele Funktionen in beiden Systemen bereits vorhanden sind. Bei einer Integration müssten in jedem Fall entweder Teile aus Triana durch deren Entsprechung aus RCE ersetzt werden, oder umgekehrt. Eine derartige Verschmelzung von zwei sich überschneidenden Systemen ist schwierig und bietet kaum mehr Nutzen gegenüber der Neuimplementierung der in einem System fehlenden Funktionen.

## 4.4 Einordnung

Alle drei Workflowsysteme sind konzeptuell ähnlich. Sie unterscheiden sich im Wesentlichen in Anwenderzielgruppe und darin, auf die Lösung welches Problems sie einen besonderen Fokus legen. Kepler zum Beispiel soll möglichst in allen Domänen und Anwendungsfällen einsetzbar sein, unabhängig davon, welche Semantik dieser von dem zugrunde liegenden System verlangt. Askalon ist auf Gridcomputing und insbesondere auf performante Durchführung von Workflows optimiert. Triana hingegen versucht middlewareagnostisch zu sein und möglichst viele unterschiedliche Systeme anzubinden. Alle Tools legen besonderen Wert auf eine graphische Benutzeroberfläche und alle nutzen – wenn auch mit unterschiedlicher Terminologie – ähnlich Konzepte, um Datenfluss zu modellieren.

Ein im Rahmen dieser Master-Thesis entwickeltes Workflowsystem kann, was Funktionsumfang und verwendete Algorithmen angeht, nicht mit diesen Systemen gleichziehen, da deren Entwicklungsvorsprung zu groß ist. Allerdings kann es sich dadurch von den genannten Projekten absetzen, dass es auf Eclipse aufbaut und damit selber komponentenorientiert entwickelt werden kann. An Kepler, Triana und Askalon wird bereits seit mehreren Jahren mit großen Entwicklerteams entwickelt. Dank des komponentenorientierten Entwicklungsansatzes in RCE, kann ein einfaches Workflowsystem, fast ausschließlich aus vorhandenen Komponenten, innerhalb weniger Monate von einer Person entwickelt werden. Der Entwicklungsansatz erlaubt außerdem, dass viele der Features, die Askalon, Triana und Kepler diesem System noch voraus haben, falls sie benötigt werden, nachgerüstet werden können. Das Grid Application Toolkit (GAT) wird zum Beispiel derzeit in einem anderen Projekt in RCE integriert und ist nach Abschluss dieser Integration auch von dem RCE-Workflowsystem nutzbar. Auf ähnliche Art ließen sich nachträglich komplexe Scheduler oder Monitoringkomponenten nachrüsten.

# Kapitel 5

# Konzept

In den Kapiteln 1 und 2 wurde dargelegt, was ein Workflowsystem ist, wofür es gebraucht wird und was die Anforderungen an ein solches System sind. In Kapitel 4 wurden Projekte vorgestellt, die sich mit diesen Problemen beschäftigen, allerdings einige Anforderungen aus dem RCE-Projekt nicht erfüllen.

In diesem Kapitel wird ein Konzept vorgestellt, wie RCE um ein Workflowsystem erweitert werden und wie dieses Workflowsystem aufgebaut sein kann.

Wie die Autoren von Askalon, Kepler und Triana festgestellt haben, sind wissenschaftliche Workflows in der Regel sehr datenzentriert und leicht als Datenfluss zu modellieren [Ludäscher u. a. (2006); Fahringer u. a. (2007); Taylor u. a. (2007)]. Aufgrund der geforderten Einfachheit des Workflowsystems erscheint es sinnvoll, auf Kontrollfluss zu verzichten und ein rein datenflussbasiertes Workflowsystem zu entwickeln. Dies ist darin begründet, dass Nutzer eines solchen Systems lediglich angeben müssen, woher die zu verarbeitenden Daten kommen und wohin sie gehen, nicht aber, wie die Daten verarbeitet werden sollen. Eine Mischung von Kontrollfluss und Datenfluss könnte für den Workflowentwickler zu komplex werden.

Ein grobes Konzept, dass einfach und zugleich hinreichend mächtig ist, lässt sich wie folgt definieren:

- Ein Workflow besteht aus Komponenten.
- Komponenten haben keinen gemeinsamen Speicher.
- Komponenten führen eine Berechnung durch.
- Komponenten erhalten Eingabedaten.
- Komponenten produzieren Ausgabedaten.
- Die Ausgabedaten einer Komponente können die Eingabedaten einer beliebigen Komponente sein. Um die Mächtigkeit ausreichend hoch zu halten, muss die Definition von Schleifen unterstützt werden. Das heißt, die Ausgabe einer Komponente kann zur Eingabe derselben Komponente oder einer anderen vorangehenden Komponente werden.
- Ein Workflow kann selber eine Komponente sein.

Dieses Konzept ist einfach umzusetzen und zu verstehen, da es folgende Eigenschaften hat:

- Komponenten sind frei von Nebeneffekten[1].
- Ein Workflow kann als gerichteter Graph dargestellt werden, in dem die Logik ausschließlich in den Knoten und nicht in den Kanten ausgedrückt wird (Knoten sind Komponenten, Kanten sind Datenwege).
- Da Komponenten nebeneffektfrei sind, gibt es wenig Anlass für Synchronisation.
- Falls Synchronisation erforderlich sein sollte, kann diese über die Datenschnittstelle implizit erfolgen[2].

Das Workflowsystem ausschließlich datenflussbasiert zu gestalten wirft die Frage auf, ob Datenfluss alleine ausreicht, um die gegebenen Anforderungen zu erfüllen. Die Anforderung die dies betrifft ist, dass Schleifen und Bedingungen realisierbar sein müssen (siehe Kapitel 2). Die Unterstützung von Schleifen hat für einen datenflussbasierten Workflow zur Folge, dass der Datenfluss nicht als azyklischer Graph dargestellt werden kann, ist aber kein grundsätzliches Problem. Bedingte Ausführungen sind durch Datenfluss nicht modellierbar. Allerdings können Daten gefiltert werden, d.h. das Auswählen von Daten, die ein bestimmtes Kriterium erfüllen, ist möglich. Über solche Filter lassen sich Datenflüsse modellieren, die vom Ergebnis her äquivalent mit bedingten Ausführungen sind. Siehe hierzu auch Abschnitt 5.5.

Eine Workflowkomponente muss also lediglich Daten über eine wohldefinierte Schnittstelle annehmen und mittels der gleichen Schnittstelle Daten ausgeben um die Anforderungen zu erfüllen. Eine einfache Schnittstelle, die hierzu geeignet ist, wird in 5.1 beschrieben.

Weiterhin gehören Wiederverwendbarkeit der Komponenten und einfache Kommunikationsmöglichkeiten zu den Anforderungen. Außerdem sollte es möglich sein, die Implementationsdetails einer Komponente zu verstecken. Alle drei Anforderungen lassen sich zwar über Java-Interfaces prinzipiell erfüllen, allerdings lässt sich, bei schlecht definierten Interfaces, oft trotzdem auf Funktionsweise und verwendeten Algorithmen schließen. Es wäre also nötig, dass es für jede beliebige Aufgabe ein von vornherein wohldefiniertes Interface existiert. Um dies zu erreichen gibt es zwei Möglichkeiten:

- Komponentenentwickler definieren die Java-Interfaces selber.
- Die Java-Interfaces werden vorgegeben.

Definieren die Entwickler ihre eigenen Interfaces, kann es sein, dass Komponenten unterschiedlicher Entwickler, die die gleiche Aufgabe erfüllen, unterschiedliche Interfaces besitzen. Aus diesem Grund wären sie dann nur schwer austauschbar. Da es aber kaum möglich ist, für alle möglichen Aufgaben vorab Java-Interfaces zu definieren, erscheint die Verwendung von Java-Interfaces zur Erfüllung der Anforderungen nicht ausreichen. Auch diese Anforderungen lassen sich durch die Verwendung von datenflussgesteuerten Workflows lösen, da es bei diesen nur notwendig ist, zu definieren, welche Daten eine Komponente annimmt und welche sie ausgibt.

Die Anforderungen können also durch datenflussbasierte Workflows erfüllt werden und legen deren Verwendung sogar nah.

[1] Vollständige Freiheit von Nebeneffekten kann nicht erreicht werden, da es zum Beispiel möglich sein muss, Ergebnisse in einer Datei zu speichern und diese zu archivieren, oder Eingabedaten aus einer Datei zu lesen. Das Schreiben oder Lesen einer Datei sind aber typische Nebeneffekte. Es können nur solche Nebeneffekte ausgeschlossen werden, die direkt eine andere Komponente beeinflussen (z.B. Zustandsveränderungen).

[2] z.B. über blockierende Leseoperationen oder Schließen der Verbindung.


Abbildung 5.1: *Darstellung von einem Kanal. A und B bezeichnen Komponenten die jeweils einen Endpunkt In und einen Startpunkt Out besitzen. Der Startpunkt Out der Komponente A ist mit dem Endpunkt In der Komponente B mit einem Pfeil verbunden. Startpunkt, Endpunkt und Pfeil bilden zusammen einen Kanal und sind daher in dieser Abbildung rot hinterlegt. Komponente A kann auf den Kanal schreiben, Komponente B kann von dem Kanal lesen. Hierzu nutzen sie ihren Start- bzw. Endpunkt.*

## 5.1 Kanäle

Da die Kommunikation mit anderen Komponenten einfach sein soll, liegt die Spezifizierung einer simplen Datenschnittstelle nahe. In den in 3.2 beschriebenen Shell-Skripten und Unix-Shells ist eine derartige Datenschnittstelle ein zentrales Element. Die zeilenweise Eingabe über Standardeingabe und Standardausgabe und die Definition von Kommunikationswegen mittels Pipes ist sowohl einfach als auch mächtig. Dieser Mechanismus lässt sich zudem einfach auf das RCE-System übertragen, da dieses bereits eine Reihe der benötigten Technologien – wie zum Beispiel die Kommunikation in verteilten Systemen – bereitstellt. Das Ergebnis dieser Überlegung nennt sich *Kanäle* oder *Channels*

Ein Kanal hat eine Startpunkt und einen Endpunkt. Daten fließen vom Startpunkt zum Endpunkt[3]. Statt wie bei Unix-Pipes die Daten zeilenweise zu verschicken, werden Kanäle allerdings typisiert. Dies bedeutet, dass ein Kanal nur Werte eines vorher festgelegten Typs verschicken kann. Als Typ ist dabei jede Java-Klasse erlaubt, die das Java-Interface *Serializable*[4] implementiert. Schließlich werden Objekte dieses festgelegten Typs einzeln verschickt.

Komponenten verschicken also Werte über Startpunkte und empfangen Daten an den Endpunkten. Eine Komponente kann über beliebig viele Start- und Endpunkte verfügen. Auch die Anzahl der Start- und Endpunkte innerhalb einer Komponente muss nicht übereinstimmen. Um mit einer anderen Komponente zu kommunizieren, müssen Werte von der sendenden Komponente einfach nur in den Startpunkt eines Kanals geschrieben werden. Zuvor muss der Startpunkt lediglich mit einem Endpunkt einer anderen Komponente verbunden worden sein (was in der graphischen Benutzeroberfläche zum Erstellen und Bearbeiten von Workflows geschieht). Die empfangende Komponente kann nun einfach über den verbundenen Endpunkt von dem Kanal lesen. Dabei muss garantiert sein, dass Werte in der Reihenfolge ankommen, in der sie abgeschickt werden. Ferner muss eine Komponente informiert werden, wenn keine neuen Eingabewerte mehr zu erwarten sind. Es muss also möglich sein, dass Kanäle geschlossen werden können. Siehe hierzu auch Abbildung 5.1.

Die folgenden Stichpunkte fassen diese Überlegungen zusammen:

[3] Aus Komponentensicht stellt ein Startpunkt also die Ausgabe und ein Endpunkt die Eingabe dar.

[4] Das RCE-System stellt mit dem RCE-Communications-Bundle einen Mechanismus zur Verfügung, der es erlaubt, Objekte, deren Klassen das Java-Interface *Serializable* implementieren, mittels verschiedener Transportprotokolle zu verschicken. Um diese Funktionalität wiederzuverwenden, ist es also nötig sich auf serialisierbare Datentypen zu beschränken.

- Komponenten tauschen Daten über Kanäle aus.
- Ein Kanal besteht aus einem Start- und einem Endpunkt.
- Ein Startpunkt eines Kanals ist die Ausgabe einer Komponente.
- Ein Endpunkt eines Kanals ist die Eingabe einer Komponente.
- Auf Kanälen müssen wohldefinierte Daten transportiert werden (Serialisierte Java-Objekt-Instanzen eines vorher definierten Typs).
- Kanäle müssen geschlossen werden, wenn sie keine neuen Daten mehr bekommen.
- Kanäle müssen die Reihenfolge der Daten, die sie transportieren, aufrecht erhalten.

## 5.2 Graphische Repräsentation

Wie in 3.2 dargestellt, gibt es bereits einige Workflowsprachen, die über eine graphische Repräsentation verfügen. Zu diesen gehören YAWL, AGWL und BPMN. YAWL verwendet Petri-Netze zur Darstellung, welche aber zur Modellierung von Kommunikationsabläufen ungeeignet[5] sind. AGWL verfügt über eine auf UML basierende Repräsentation. UML-Aktivitätsdiagramme werden allerdings schon bei wenigen Akteuren schnell komplex und unübersichtlich und eignen sich daher eher für einfache, schematische Darstellungen. BPMN ist eine rein graphische Sprache, die sehr mächtig aber auch sehr umfangreich ist.

Obwohl diese drei Repräsentationen nicht uneingeschränkt für den gegebenen Anwendungsfall geeignet sind, kann es sinnvoll sein, auf eine von ihnen aufzubauen, da sie alle standardisiert und weit verbreitet sind, was sich positiv auf den Einarbeitungsaufwand vieler Nutzer auswirken kann. Zusätzlich existieren bereits graphische Editoren, die lediglich angepasst und in RCE integriert werden müssten.

Während die oben genannten Kritikpunkte an Petri-Netzen (Kontrollfluss steht im Vordergrund) und UML (bei mehreren Akteuren schnell unübersichtlich) nur schwer auszuräumen sind, ist der enorme Umfang des BPMN Sprachschatzes nur ein kleines Problem, da sich das Konzept der Kommunikationskanäle und Komponenten mit einer sehr kleinen Untermenge der BPMN-Sprachelemente ausdrücken lässt.

Eine ausreichende Untermenge von BPMN besteht aus den Elementen: *Pool*, *Task* und *Message Flow Edge* (siehe Tabelle 5.1) In der BPMN stellt der *Pool* den Rahmen oder Kontext dar, in dem einzelne *Activities* ausgeführt werden. Ein *Task* ist die simpelste *Activity* und kann zum Beispiel mittels *Message Flow Edges* verbunden werden. Diese Elemente können wie folgt auf den RCE-Anwendungsfall abgebildet werden: Jede Komponente wird durch einen *Pool* dargestellt. Jeder Start- und Endpunkt, den eine Komponente zur Verfügung stellt, wird durch einen *Task* dargestellt. Die Verbindung eines Startpunktes mit einem Endpunkt durch eine *Message Flow Edge* repräsentiert einen Kanal. Diese Abbildung von BPMN-Elementen auf das vorgestellte Konzept ist in Tabelle 5.1 zusammengefasst.

[5] Es ist zwar möglich Datenfluss mit Petri-Netzen zu modellieren, allerdings enthalten Petri-Netze immer auch Kontrollflusselemente. Diese sind nur sehr schwer von den Datenflüssen zu unterscheiden. Daher würde Datenflussmodellierung mit Petri-Netzen ein Verständnis der Petri-Netz-Theorie vom Nutzer verlangen.

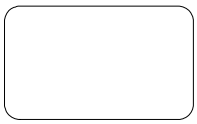
| Symbol | BPMN Bedeutung | Bedeutung in diesem Workflowsystem |
|---|---|---|
|  | *Pool* - Repräsentiert einen Teilnehmer in einem Prozess. Container für andere BPMN-Elemente (wie zum Beispiel *Task*). | Komponente |
|  | *Task* - Repräsentiert eine einzelne Aufgabe in einem Prozess. | Start- und Endpunkt. |
|  | *Message Flow Edge* - repräsentiert einen Nachrichtenfluss zwischen zwei *Tasks* in Pfeilrichtung. | Datenfluss/Kanal |

Tabelle 5.1: Abbildung von BPMN-Elementen (Grafiken aus [OMG (2006)]) auf das vorgeschlagene Workflowsystem und seine Elemente.

Zu Eclipse gehört seit Version 3.3 die Eclipse Service Tools Platform (STP). Diese beinhaltet einen graphischen Editor für BPMN-Diagramme, der auch beliebige Zusatzinformationen zu jedem graphischen Element in Form von *Annotations* erlaubt. Mit diesen Annotations ist es möglich, von BPMN-Elementen repräsentierte Komponenten zu parametrisieren. Dies führt zusammen mit den vorangegangenen Überlegungen dazu, dass die beschriebene BPMN-Untermenge zur graphischen Repräsentation von Workflows genutzt wird und eine Benutzerschnittstelle auf Basis von STP in RCE integriert wird.

## Entkopplung

Die graphische Benutzerschnittstelle soll vom Rest des Systems und insbesondere von den Komponenten unabhängig sein. Es soll mit der graphischen Benutzerschnittstelle möglich sein, Workflows zu erstellen und zu bearbeiten, ohne dass diese gestartet werden müssen. Die Informationen darüber, welche Start- und Endpunkte eine Komponente besitzt, muss zugänglich sein, ohne dass die Komponente dafür instantiiert oder eine bestimmte Klasse geladen werden muss. Daher ist es nicht möglich, Komponenten ein Java-Interface zu geben, das es erlaubt, die Details über Start- und Endpunkte abzufragen. Diese Informationen müssen in einer separaten Beschreibungsdatei abgelegt werden, wo sie dem System jederzeit zur Verfügung stehen. Eclipse-Plugins verfügen über die Beschreibungsdatei *plugin.xml*, worin Informationen über ein Plugin abgelegt werden. Da RCE-Methoden immer als Eclipse-Plugins realisiert sind und die Workflowkomponenten RCE-Methoden sein sollen, verfügen auch die Workflowkomponenten über diese Datei. Daher können die benötigten Informationen über Start- und Endpunkte dort abgelegt werden.

## 5.3 Prozesssteuerung

Nachdem ein Workflow definiert ist, muss es möglich sein, diesen Workflow zu starten. Hierzu müssen die einzelnen Komponenten auf den dafür vorgesehenen RCE-Instanzen gestartet und die Kommunikationskanäle konfiguriert werden. In diesem Kapitel wird vorgestellt, wie ein Workflow gestartet und überwacht werden kann.

### 5.3.1 Scheduling

In einem verteilten Workflowsystem ist es wichtig, vorhandene Ressourcen so gut es geht auszunutzen. Insbesondere eine Lastverteilung ist sehr wünschenswert. Hierzu sind allerdings eine Reihe von Informationen über die zu startenden Methoden notwendig. Insbesondere wichtig sind hier Rechenaufwand und Kommunikationsaufwand. Ziel eines Schedulingverfahrens wäre es dann, Komponenten, die viel miteinander kommunizieren, auf dem gleichen Rechner oder auf Rechnern im gleichen Netzwerk zu starten. Rechenintensive Komponenten sollten dabei auf verschiedenen Rechnern laufen.

Die Menge dessen, was eine Komponente ausgibt, steht immer im Verhältnis zu dem, was die Komponente als Eingabe erhält. Eine Komponente $K_1$ mit den Endpunkten $a$ und $b$ sowie dem Startpunkt $c$, der die Formel

$$a_n + b_n = c_n \tag{5.1}$$

berechnet, würde zum Beispiel Bei Eingabe von $n$ Werten für $a$ und $n$ Werten für $b$ ein Eingabevolumen von

$$|a| + |b| = n + n = 2n \tag{5.2}$$

und ein Ausgabevolumen von

$$|c| = n \tag{5.3}$$

haben. Es gilt also:

$$Ausgabevolumen = \frac{1}{2} Eingabevolumen \tag{5.4}$$

Diese Komponente würde das Kommunikationsvolumen also reduzieren.

Eine Komponente $K_2$ mit dem Endpunkt $m$ und dem Startpunkt $p$, die von einer gegebenen Menge $M$ die Potenzmenge $P_M$ berechnet und alle Elemente einzeln ausgibt, würde hingegen das Kommunikationsvolumen stark vergrößern, da

$$|M| = n \Rightarrow |P_M| = 2^n \Rightarrow |M| \ll |P_M| \tag{5.5}$$

. Daraus ergibt sich:

$$Ausgabevolumen = 2^{Eingabevolumen} \tag{5.6}$$

.

Für einen Scheduler wäre es folglich theoretisch sinnvoll, $K_1$ auf dem gleichen Rechner laufen zu lassen wie die Komponenten, die die Eingabewerte für $K_1$ bereitstellen, während es für $K_2$ sinnvoller wäre, auf dem gleichen Rechner zu laufen, auf dem auch die empfangende, weiterverarbeitenden Komponenten laufen. Allerdings ist der Rechenaufwand noch nicht in Betracht gezogen worden. Der Aufwand, den $K_2$ zu betreiben hat ist relativ hoch. Führt die weiterverarbeitende Ressource selbst aufwändige Berechnungen auf jedem Element

von $P_M$ durch, kann es durchaus sinnvoller sein, beide Komponenten auf unterschiedlichen Rechnern laufen zu lassen.

Ein effektives Scheduling würde also sowohl eine Reihe von Informationen über die Komponenten, als auch Informationen über die Netzwerktopologie und die Rechenleistung der beteiligten Rechner benötigen. Außerdem ist es in einem RCE-System durchaus möglich, dass auf einer RCE-Instanz manuell weitere Methoden gestartet werden. Dies könnte für einen Scheduler unvorhergesehene Lasten erzeugen und das Scheduling so behindern. Weiterhin ist es möglich, dass einzelne RCE-Methoden aufgrund von Lizenzbedingungen nur auf bestimmten Rechnern laufen dürfen. Was den Kommunikationsaufwand angeht, so ist das vorhin Erwähnte zwar für die Kanäle wahr, allerdings kann eine Komponente durchaus auch auf Datenbanken oder Dateien zugreifen. In diesem Fall kann über den Kommunikationsaufwand ebenfalls keine allgemeingültige Aussage getroffen werden.

Aufgrund des Umfangs und der Komplexität dieser Probleme, kann in dieser Master-Thesis nur ein sehr einfacher Schedulingmechanismus realisiert werden[6]. Dieser einfache Schedulingmechanismus wird folgendermaßen funktionieren:

Auf einen eigenen Workflowscheduler wird verzichtet. Workflowkomponenten werden schon in der GUI mittels Annotations[7] auf den Start auf einer bestimmten RCE-Instanz eingestellt. Wird diese Information weggelassen, wird der RCE-Standardmechanismus zum Starten von Methoden verwendet. Bei diesem ist es derzeit nicht möglich Vorhersagen darüber zu treffen, auf welcher Instanz eine RCE-Methode gestartet wird, falls keine bestimmte Instanz gefordert wurde. Wird später ein Schedulingmechanismus benötigt, der Lastausgleich und Performanceoptimierung betreibt, kann dies in den Standardmechanismus integriert werden, so dass nicht nur das Workflowsystem von dieser Neuerung profitiert. Eine Minimierung des Kommunikationsaufwands kann lediglich bei sehr simplen Komponenten erfolgen (z.B. Filter-Komponenten, die einen Wert weiterleiten, wenn er ein bestimmtes Kriterium erfüllt oder ansonsten unterdrücken). Solche simplen kommunikationsreduzierenden Komponenten, die wenig Last erzeugen, könnten immer auf dem System gestartet werden, auf dem die vorgeschaltete Komponente läuft.

Sobald eine Komponente nicht mehr weiterarbeiten kann, weil keine Eingabewerte vorliegen, kann sie mit Hilfe des Java-*wait/notify*-Protokolls schlafen gelegt und wieder aufgeweckt werden, sobald wieder Daten vorhanden sind. Dies wird über die Kanäle realisiert, indem das Lesen von einem Endpunkt als blockierende Operation implementiert wird. Eine nicht blockierende Leseoperation wird allerdings dennoch zur Verfügung gestellt, damit Komponenten, die auch wenn keine neuen Daten vorliegen weiterarbeiten können, nicht unnötig behindert werden.

### 5.3.2 Monitoring

Das RCE-System verfügt über eine Monitoringansicht, die allerdings derzeit noch relativ beschränkt ist. Sie zeigt lediglich an, welche Komponenten im aktuellen System gestartet sind. Details wie zum Beispiel die Laufzeit und die Zeit, die eine Methode auf Eingaben wartet, sind derzeit nicht verfügbar. Ebenso ist keine

[6] Die Entwickler von Askalon beschäftigen sich intensiv mit den Problemen beim Scheduling von Workflows und haben verschiedene Lösungsansätze untersucht [Wieczorek u. a. (2005)]

[7] Gemeint sind hier nicht Java-Annotations oder BPMN-Annotations sondern die im STP-BPMN-Modeler – der Software auf der die GUI basieren wird – verwendbaren GEF (Graphical editing Framework) *EAnnotations* [Eclipse-Foundation (2008a)].

Lastanalyse für die einzelnen RCE-Instanzen verfügbar. Derartige Informationen sind als äußerst wichtig für ein Workflowsystem einzustufen, da sie bessere Scheduling Entscheidungen erlauben und Anwendungsentwicklern Hinweise geben können, wie sie ihre Workflowkomponenten optimieren können. Da die notwendigen Techniken allerdings noch nicht in RCE integriert sind und komplexes Scheduling für die Entwicklung im Rahmen dieser Master-Thesis verworfen wurde, wird auch auf Monitoring verzichtet[8].

## 5.4 Komponenten

Nachdem in den vorherigen Abschnitten erläutert wurde, wie Komponenten miteinander kommunizieren und wie man Komponenten darstellen kann, folgt nun eine Definition dessen, wie eine Komponente für das vorgeschlagene Workflowsystem aufgebaut sein muss.

- Eine Komponente ist eine RCE-Methode. Das bedeutet, dass eine Komponente auch ein Eclipse-Plugin ist.
- Eine Komponente verfügt über Start- und Endpunkte zum Aufbau von Kommunikationskanälen.
- Über welche Start- und Endpunkte eine Komponente verfügt, muss in der zugehörigen *plugin.xml*-Datei spezifiziert werden. Hierzu muss das plugin.xml-Schema angepasst werden (siehe Listing A.5).

## 5.5 Modellierung von Prozessen

Um zu demonstrieren, dass es mit dem erarbeiteten Konzept möglich ist, Workflows zu erstellen und komplexe Prozesse zu modellieren, wird dies in diesem Abschnitt an einigen Beispielen vorgestellt.

Der erste Workflow ist in Abbildung 5.2(a) dargestellt (bzw. vereinfacht in Abbildung 5.2(b)) und erlaubt die Berechnung der Fibonaccizahlen[9]. Hierzu werden folgende Komponenten benötigt:

- Ein Addierer ($Add$) mit zwei Endpunkten ($a, b$) und einem Startpunkt ($c$). Der Addierer berechnet die Formel $a + b = c$. Die Ausgabe lautet also $c_n = a_n + b_n$.
- Zwei Bootstrap-Komponenten ($Boot$) mit einem Endpunkt ($in$) und einem Startpunkt ($out$), sowie einer Annotation ($first\_value$). Die Bootstrap-Komponente liefert Werte nach dem folgenden Schema $out_0 = first\_value; out_n = in_{n-1} \forall n \geq 1$.
- Eine Kopier-Komponente ($Tee$)[10] mit einem Endpunkt ($in$) und zwei Startpunkten ($a, b$). Die Ausgaben ergeben sich als $a_n = in_n, b_n = in_n$.
- Eine Rotationskomponente ($Cycle$) mit einem Endpunkt ($in$) und zwei Startpunkten ($a, b$). Die Ausgaben ergeben sich als $a_n = in_{n+1}, b_n = a_n \forall n \geq 0$.

[8] Beide Funktionalitäten können dank dem komponentenorientierten Design von RCE später ohne großen Aufwand nachgerüstet werden.

[9] Die Fibonacci-Folge $(f_0, f_1, \ldots)$ ist definiert als $f_0 = 0; f_1 = 1; f_{n+2} = f_{n+1} + f_n \forall n \geq 0$. Die ersten 10 Fibonaccizahlen lauten $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$ [Knuth (1997)]

[10] Dieser Komponentenname ist angelehnt an das Unix-Kommando *tee* welches die Eingabe gleichzeitig auf der Standardausgabe und in einer Datei ausgibt, was in etwa das gleiche ist, was diese Kopier-Komponente tut.

Die Fibonaccifolge wird auf dem Startpunkt $b$ der Komponente $Tee$ ausgegeben.

Dieser Workflow zeigt, dass sich mit dem Konzept problemlos Schleifen erzeugen lassen. Der gesamte Workflow ist exemplarisch in Listing A.1 dargestellt.

Der zweite Workflow ist in Abbildung 5.3(a) (vereinfacht in Abbildung 5.3(b)) dargestellt und ist eine iterative Variante des Euklidischen-Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier Zahlen. Es werden folgende Komponenten eingesetzt:

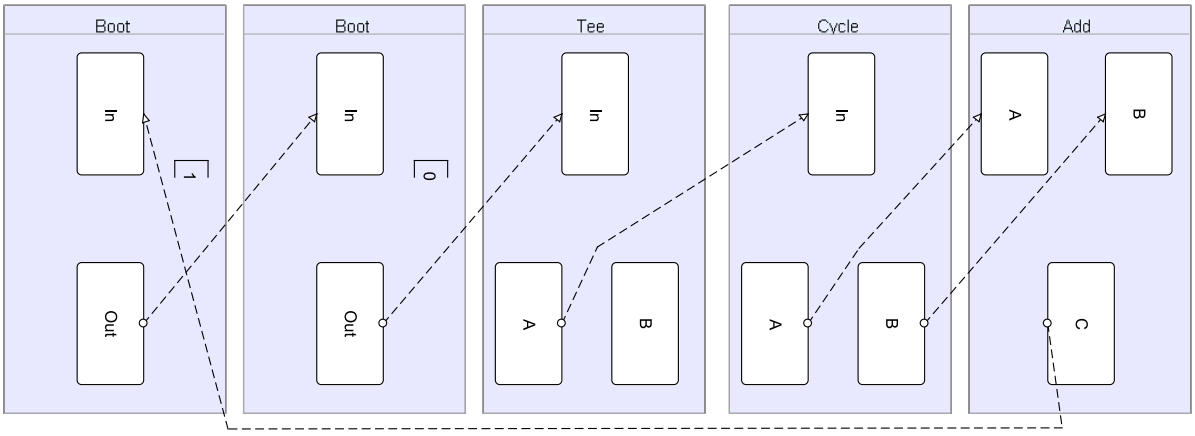
- Zwei Bootstrap-Komponenten ($Boot$) wie im vorherigen Workflow zur Berechnung der Fibonaccizahlen. Diesmal mit den Zahlen $a$ und $b$ initialisiert.
- Zwei Kopier-Komponenten ($Tee$) wie im vorherigen Beispiel.
- Eine Modulo-Komponente ($Modulo$) mit zwei Endpunkten $(a, b)$ und einem Startpunkt $(c)$. Die Komponente berechnet die Formel $a \bmod b = c$. Die Ausgabe lautet also $c_n = a_n \bmod b_n$.
- Eine Kontroll-Komponente ($Iterator$) mit zwei Endpunkten $(a, b)$ und zwei Startpunkten $(result, next)$, deren Ausgabe sich nach den Formeln $result = a_n$, falls $b_n = 0$ und $next_n = b_n$, falls $b_n \neq 0$ berechnet. Der Pseudocode für diese Komponente ist in Listing A.2 zu sehen.

Dieser Workflow demonstriert, dass es mit dem System möglich ist, Verzweigungen abzubilden. Die $Iterator$-Komponente ist im Prinzip eine Komponente, die eine bestimmte Bedingung auswertet und Eingaben auf Grund dieser Bedingung filtert. Wie diese Funktionalität in Java ausgedrückt werden kann, ist in Listing A.2 dargestellt. Eine Komponente für allgemeine Bedingungen ($filter$) könnte ähnlich definiert werden. Die Eingaben, für die die Bedingung wahr ist, werden auf dem Startpunkt $a$ ausgegeben, die, für die die Bedingung nicht wahr ist, werden auf dem Startpunkt $b$ ausgegeben. Eine solche Komponente müsste die Bedingung in Form einer Annotation übergeben bekommen (zum Beispiel als Groovycode[11]). Eine Implementation einer solchen Filter-Komponente zeigt Listing A.3.

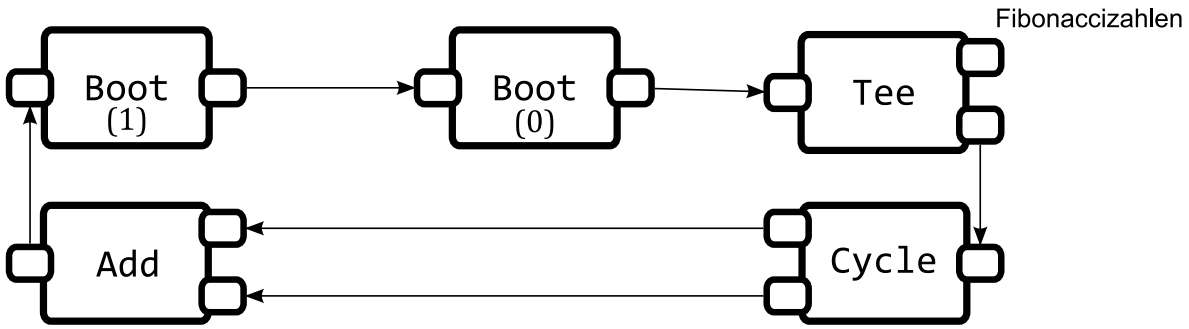
Der dritte Workflow ist in Abbildung 5.4(a) (vereinfacht in 5.4(b)) dargestellt. Hierbei handelt es sich um ein Hidden-Markov-Modell als Workflow ausgedrückt. Als Komponente kommt lediglich eine Markov-Element-Komponente zum Einsatz (Ausgenommen der bereits bekannten Boot-Komponente). Jedes dieser Markov-Elemente repräsentiert einen Hidden-State in einem Hidden-Markov-Modell Diese hat einen Endpunkt $(in)$ und zwei Startpunkte $(visible, next)$. Die Ausgaben sind wie folgt definiert: $visible_n = f_{visible}()$, falls $in_n = 0$, $next_n = in_n - 1$, falls $n > 0$ $next_n = f_{next}()$, falls $in_n = 0$. $f_{visible}$ ist hierbei eine Funktion, die Ausgaben mit deren Emissionswahrscheinlichkeiten erzeugt und $f_{next}$ ist eine Funktion, die die Anzahl der Sprünge ausgibt, die nötig ist, um zu dem Hidden-Markov-Element zu gelangen, das den folgenden Hidden-State repräsentiert. Die Ausgabe ist abhängig von den für das Modell definierten Zustandsübergangswahrscheinlichkeiten. Jedes Markov-Element benötigt also zwei Annotations, in denen Emissionswahrscheinlichkeiten und Zustandsübergangswahrscheinlichkeiten hinterlegt sind.

Dieser dritte Workflow demonstriert, dass es mit dem Workflowsystem auch möglich ist, komplexere Probleme abzubilden, und dass die graphische Repräsentation durchaus noch einfach sein kann.

[11] Groovy ist eine Skriptsprache, die auf der Java Virtual Machine läuft und genutzt werden kann um Java-Anwendungen zu dynamisieren [Codehaus-Foundation (2008)]. Groovy wird in RCE eingesetzt.

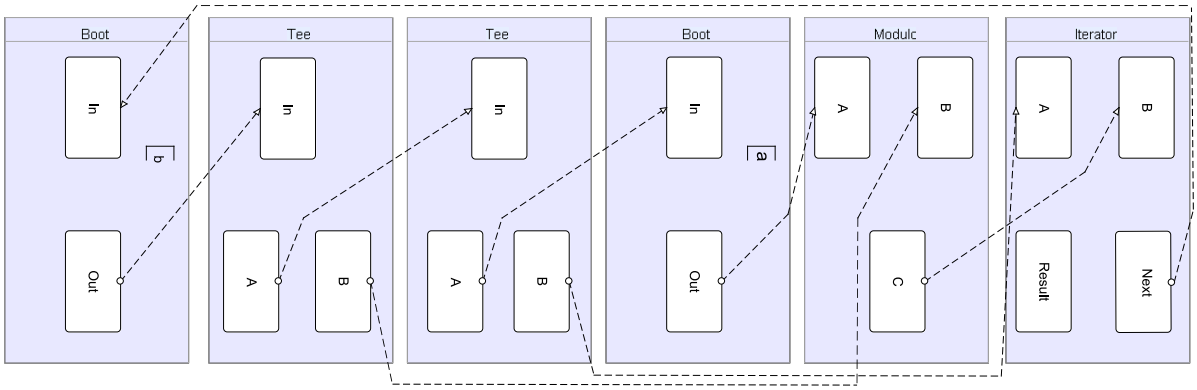



(a) Darstellung mit Standard BPMN-Elementen

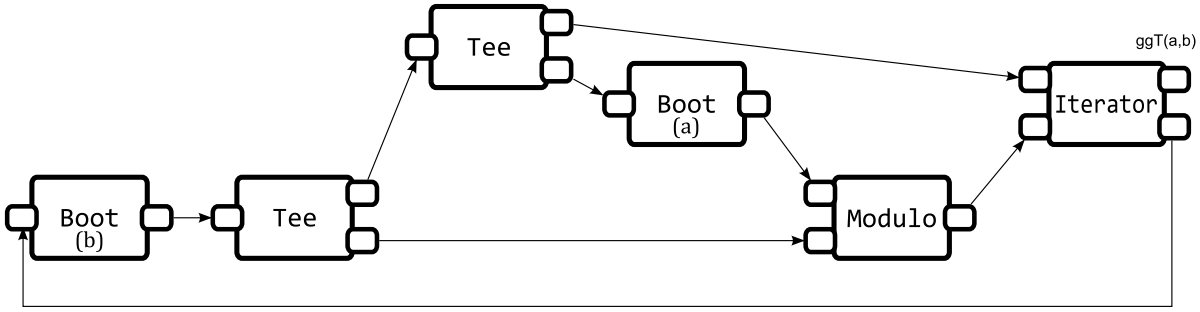



(b) vereinfachte Darstellung

Abbildung 5.2: *Workflow zur Berechnung der Fibonaccizahlen*

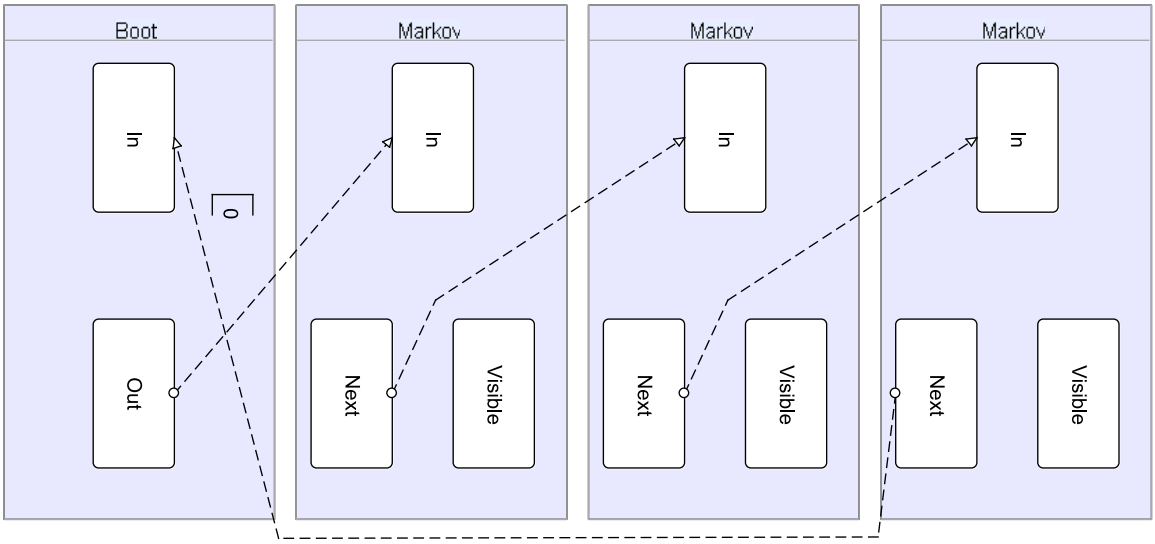



(a) Darstellung mit Standard BPMN-Elementen

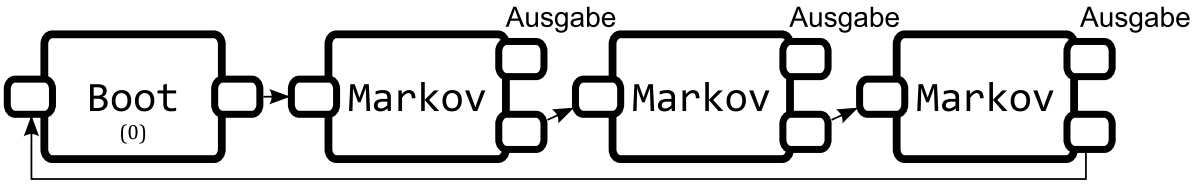



(b) vereinfachte Darstellung

Abbildung 5.3: *Euklidischer Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier Zahlen als Workflow*




(a) Darstellung mit Standard BPMN-Elementen




(b) vereinfachte Darstellung

Abbildung 5.4: *Hidden Markov Modell mit 3 Hidden States als Workflow*

# Kapitel 6

# Realisierung

Die im Rahmen der Thesis entwickelte Software umfasst mit Channelservice, Workflowservice und Workflow-GUI mehrere Eclipse-Plugins, die unabhängig voneinander arbeiten können. Alle nutzen jedoch Eclipse-Plugins, die zu dem RCE-System gehören.

In diesem Kapitel wird die Implementierung des vorgestellten Konzepts beschrieben. Der erste Teil des Kapitels beschäftigt sich mit der Implementierung des Workflowsystems und seiner Bestandteile, der zweite Abschnitt beschäftigt sich mit der Implementierung der Benutzerschnittstelle.

## 6.1 Workflow und Kanäle

Kanäle und Komponenten bilden zusammen den Kern des Workflowsystems. Kanäle bestehen, wie bereits in Abschnitt 5.1 beschrieben aus Start- und Endpunkten, die jeweils die Aus- bzw. Eingabe einer Komponente darstellen. Abbildung 6.1 stellt die implementierten Klassen in einem UML-Klassendiagramm dar. Die einzelnen Klassen sind in den folgenden Abschnitten näher erläutert.

In den folgenden Abschnitten wird ebenfalls erläutert, wie eine Workflowbeschreibung ausgeführt wird. Die hierzu entwickelten Klassen sind in Abbildung 6.2 als UML-Klassendiagramm dargestellt.

### 6.1.1 Start- und Endpunkte

Es existieren Klassen für Start- und Endpunkte. Außerdem existiert eine Klasse ValueBuffer, die Werte für Start- und Endpunkte zwischenspeichert und sicherstellt, dass Werte immer in der Reihenfolge gelesen werden, in der sie geschrieben wurden. Wird ein Wert in einen Startpunkt geschrieben, so wird diesem ein Zählwert $V$ (beginnend mit $V = 0$) zugeordnet. Die Zählwerte sind auf jedem Kanal eindeutig. Die an einem Endpunkt ankommenden Werte werden von dem ValueBuffer auf einem Heap zwischengespeichert, der so sortiert ist, dass immer der Wert mit dem niedrigsten Zählwert (der älteste Wert) als oberstes auf dem Heap liegt. Zusätzlich werden im ValueBuffer folgende Werte gespeichert:

- $v_{abholbar}$ Der niedrigste Zählwert eines noch nicht abgeholten, gepufferten Wertes.
- $v_{neuster}$ Der höchste Zählwert eines noch nicht abgeholten, gepufferten Wertes.

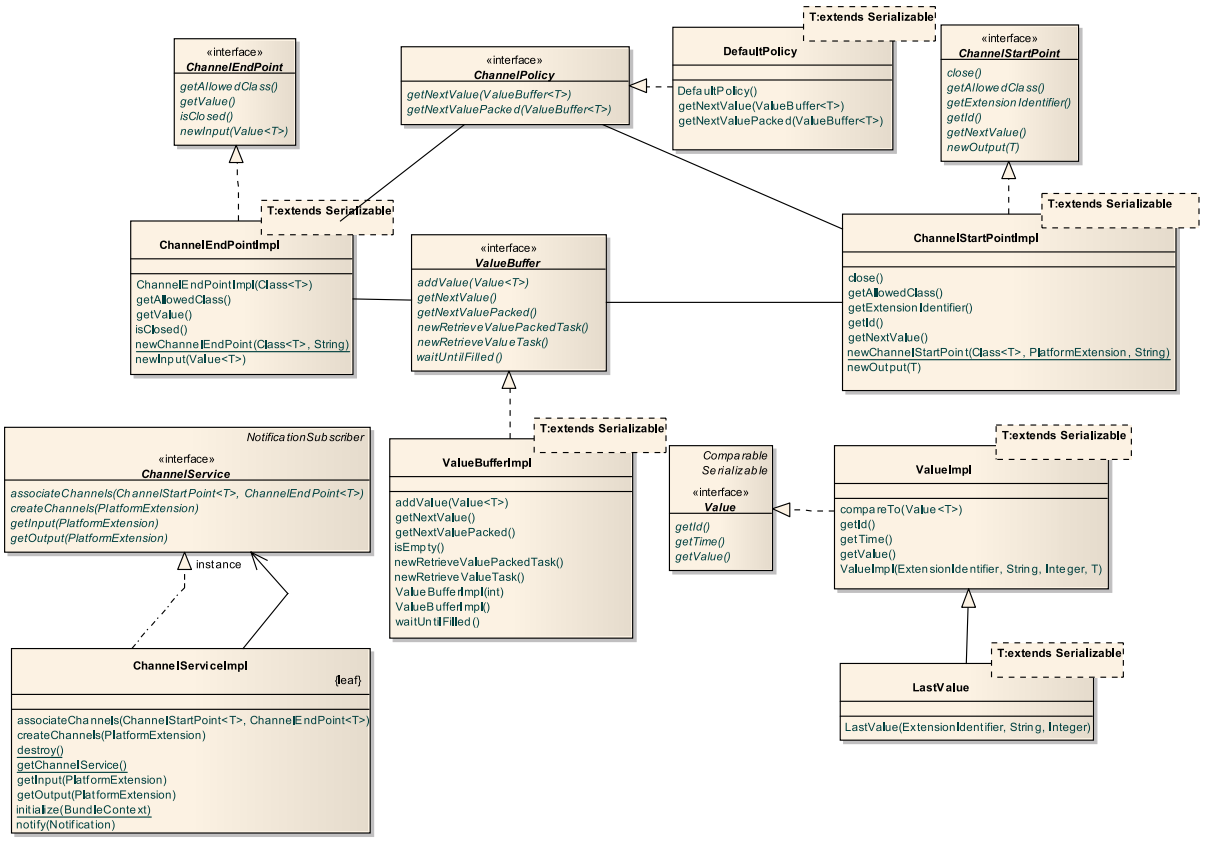



Abbildung 6.1: *UML-Klassendiagramm zu Kanälen*

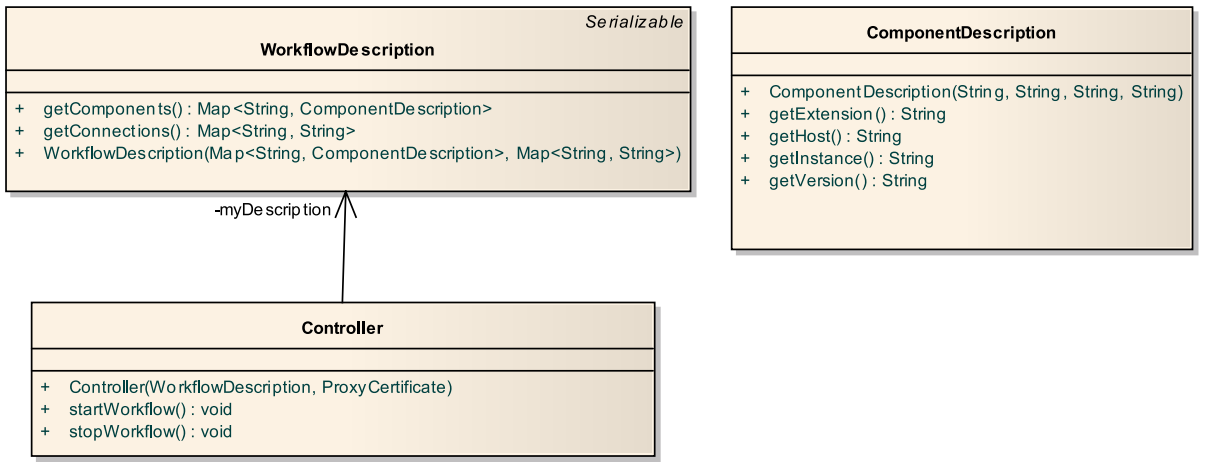



Abbildung 6.2: *UML-Klassendiagramm zum Workflowcontroller*

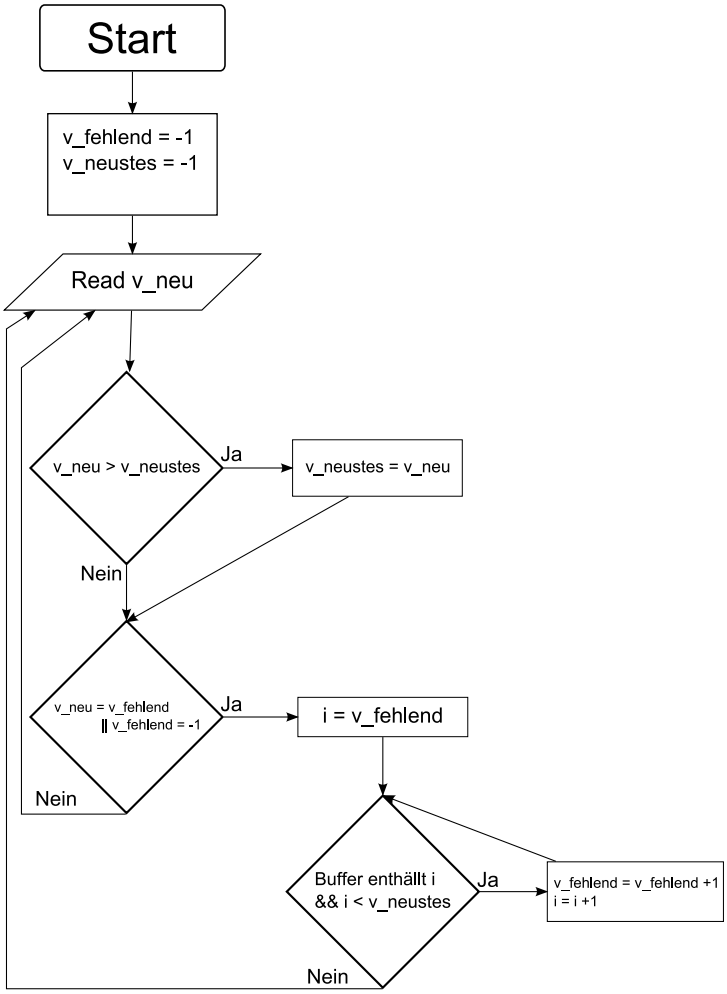



Abbildung 6.3: *Flussdiagramm zur Darstellung des Bufferordnungs-Algorithmus*

- $v_{fehlend}$ Der niedrigste Zählwert eines nicht gepufferten Wertes.

Die Anzahl der abholbaren Werte in einem ValueBuffer ergibt sich als

$$V_{fehlend} - V_{abholbar} \tag{6.1}$$

Wenn $V_{neu}$ gleich $V_{fehlend}$ ist, so wird $V_{fehlend}$ neu ermittelt. Der Algorithmus ist in Abbildung 6.3 als Flussdiagramm dargestellt.

Dieser Mechanismus kann zwar theoretisch aufwändig werden, wenn sich Werte im Netzwerk oft überholen, ist dann allerdings auch notwendig. Falls Überholungen schon durch das Netzwerkprotokoll verhindert werden, ist der Aufwand ziemlich gering, da pro Wert lediglich eine Lese-/Schreiboperation auf einem Heap notwendig wird. Diese hat die Komplexität $O(n * logn)$ und $n$ wird bei typischen Workflows lediglich im Bereich 10 bis 100 liegen. Bei diesen Größenordnungen ist die Geschwindigkeit des Netzwerkes eher von Belang als die Verzögerung durch ein Zwischenspeichern. Sollte die Reihenfolge der Werte anderweitig sichergestellt oder unwichtig sein, kann der ValueBuffer mittels eines Policy-Mechanismus ausgeschaltet werden.

Da die Namen Start- und Endpunkt sich auf einen Kanal beziehen (Anfang und Ende eines Kanals) sind die Begriffe für Komponentenentwickler ungünstig, da dieser unter einem Startpunkt eine Eingabe und unter

einem Endpunkt eine Ausgabe erwarten könnten, wenn er die Begriffe auf die Komponente bezieht. Tatsächlich ist es aber genau umgekehrt. Der Startpunkt eines Kanals nimmt die Ausgabe einer Komponente an und der Endpunkt eines Kanals liefert die Eingabe einer Komponente. Daher sind die Klassen die Start- und Endpunkt implementieren für den Komponentenentwickler verborgen. Es existieren, für Komponentenentwickler nutzbare Klassen, *Input* und *Output*, die den Sachverhalt für diese verständlich darstellen und die die Start- und Endpunkt Klassen nutzen.

### 6.1.2 Channelservice

Der Channelservice ist ein OSGi-Service, der auf jeder RCE-Instanz läuft. Wie in 5.2 beschrieben, werden die Informationen darüber, welche Kanäle eine Komponente besitzt, zur Entkopplung von Komponenten und Workflow-GUI in der *plugin.xml*-Datei des Eclipse-Plugins benötigt. Um zu verhindern, dass die Start- und Endpunkte sowohl in dieser Datei als auch im Java-Quellcode der Komponente angelegt werden müssen - was eine typische Fehlerquelle wäre, da eine identische Information an zwei unterschiedlichen Stellen in unterschiedlichen Sprachen konsistent gehalten werden muss - sorgt der Channelservice dafür, dass eine Komponente immer genau die Start- und Endpunkte bekommt, die in ihrer *plugin.xml* angegeben wurden[1]. Abgesehen davon ist der Channelservice auch noch dafür zuständig, Start- und Endpunkte miteinander zu verbinden. Hierzu stellt er Methoden bereit, die zum Beispiel von einem Workflowcontroller genutzt werden können. Dieses verbinden entspricht dem Aufbauen oder Öffnen eines Kanals. Produziert eine Komponente keine Daten für einen Kanal mehr, kann sie dies über eine dafür vorgesehene *close*-Methode des Startpunktes signalisieren. Hierdurch wird der Kanal geschlossen und von dem Endpunkt können nur noch die Werte gelesen werden, die bereits gepuffert sind[2]. Listing A.5 zeigt einen Auszug aus der EXSD (Eclipse XML Schema Definition) für die *plugin.xml*-Datei einer Workflowkomponente. Einige für diese Thesis irrelevante Details wurden aus dem Listing entfernt. Listing A.6 zeigt eine Beispiel-*plugin.xml*, für die in Abschnitt 5.5 vorgestellte $Add$-Komponente (Endpunkte $a$ und $b$ sowie ein Startpunkt $c$).

### 6.1.3 Workflowcontroller

Der Workflowcontroller ist eine Klasse, die zum Starten eines Workflows genutzt wird und den laufenden Workflow als Gesamtes verwaltet. Startet ein Benutzer einen Workflow, erhält er einen solchen Workflowcontroller, der es ihm erlaubt mit dem Workflow zu interagieren. Momentan ist die Interaktion allerdings auf das Stoppen des Workflows beschränkt. Ein Workflowcontroller wird mit Hilfe einer Workflowbeschreibung erstellt. Diese Workflowbeschreibung enthält Informationen darüber, welche Komponenten Teil des Workflows sind, wo diese laufen sollen und wie ihre Start- und Endpunkte zu Kanälen verbunden werden sollen. Sie wird mittels eines Parsers aus den von der GUI generierten XML-Repräsentationen der Workflowdiagramme erzeugt. Die Komponenten werden von dem Controller instantiiert und referenziert, dann werden die Start- und Endpunkte der Beschreibung entsprechend zu Kanälen verbunden. Schließlich werden alle Komponenten gestartet. Listing

[1] Dies läuft so, dass der Channelservice die *plugin.xml*-Dateien der Komponenten ausliest und entsprechend der Beschreibung Start- und Endpunkte instantiiert. Die Komponente kann sich diese Instanzen dann beim Channelservice abholen.

[2] Öffnen und Schließen von Kanälen ist also konzeptionell unterschiedlich. Das Aufbauen eines Kanals ist Aufgabe des Workflowentwicklers/Workflowsystems und das schließen des Kanals ist Aufgabe des Komponentenentwicklers bzw. der Komponente

A.4 stellt die Arbeitsweise des Controllers als Pseudocode dar.

## 6.2 Benutzerinterface

Neben dem im vorherigen Abschnitt vorgestellten Basissystem ist ein Workflowsystem auch auf eine intuitive Benutzeroberfläche angewiesen. Dieser Abschnitt beschäftigt sich mit der Realisierung der Oberfläche für dieses Workflowsystem. Hierbei wurden hauptsächlich vorhandene Komponenten wiederverwendet.

### 6.2.1 Workflow-Editor

Der Workflow-Editor dient zur graphischen Definition und Repräsentation von Workflows. Bei dem Workflow-Editor handelt es sich um eine modifizierte Version des BPMN-Modeler aus dem Eclipse-STP-Projekt. In Zusammenarbeit mit den BPMN-Modeler-Entwicklern wurden Schnittstellen und Mechanismen zur Modifikation des BPMN-Modelers entworfen und zur Anpassung an die Bedürfnisse eines Workflow-Editors genutzt.

In der modifizierten Version wurde im Wesentlichen der Umfang an zur Verfügung stehenden graphischen Elementen auf die in Abschnitt 5.2 beschriebene Untermenge der BPMN-Elemente beschränkt. Workflowkomponenten werden wie in Tabelle 5.1 beschrieben dargestellt. Die nötigen Informationen wie Bezeichner einer Komponente oder Namen der Hosts, auf denen diese laufen sollen, werden als Annotations angegeben und gespeichert.

Zur vollständigen Integration in RCE reicht dies allerdings noch nicht aus. Hierzu fehlt noch eine Möglichkeit die vorhandenen Komponenten direkt über eine Palette im Workflow-Editor zu einem Workflow hinzuzufügen. Diese Palette existiert zwar und der Modeler verfügt auch über einen Mechanismus mit dem die nötigen Palettenelemente erzeugt werden können und die Palette gefüllt werden kann. Allerdings ist die RCE-Komponente die Informationen in der notwendigen Form zur Verfügung stellt, um daraus die Palettenelemente zu erzeugen, noch nicht funktionsfähig. Diese Komponente gehört allerdings zum RCE-Basissystem und wird in einem anderen Projekt entwickelt. Aus diesem Grund ist der Workflow-Editor noch nicht von praktischem Nutzen. Der Aufwand, der noch erforderlich ist, um diesen letzten Integrationsschritt zu vollziehen, ist allerdings sehr niedrig und kann innerhalb weniger Tage vollzogen werden, sobald die benötigte RCE-Komponente fertiggestellt ist.

### 6.2.2 Workflowverwaltung

Verwaltung von Workflows beinhaltet die folgenden Unterbereiche:

- Speichern und Auffinden (erfordert serialisierbare Repräsentation)
- Starten und Stoppen
- Überwachen

Für alle drei Punkte werden vorhandene RCE-Plugins eingesetzt. Da es sich bei den vom Editor erstellten Beschreibungen um XML-Dokumente handelt, können diese problemlos mit jedem beliebigen Datenmanagementprogramm verwaltet werden (Die Entwicklung einer eigenen Serialisierungsmöglichkeit für die Workflowdiagramme ist also nicht nötig). Zu den Standardkomponenten von RCE gehört eine umfangreiche Datenma-

nagementlösung, die das Speichern und Versionieren von Daten erlaubt. Suchfunktionen sind ebenso enthalten wie ein Rechtemanagement. Außerdem erlaubt es Replikationen und kann mit einer Reihe von verschiedenen Storage-Backends zusammenarbeiten.

Zum Starten, Stoppen und Überwachen der Workflows wird der RCE-Extension-Browser genutzt. Dieser zeigt alle im System vorhandenen RCE Methoden an und erlaubt es, diese zu starten. Um einen Workflow zu starten existieren mehrere Möglichkeiten. Zum einen kann man eine RCE-Methode implementieren, die einen bestimmten Workflow aus dem Datenmanagement aussucht und startet, oder man kann eine RCE-Methode implementieren, die vom Benutzer eine Angabe erfordert, welchen Workflow diese starten soll. Es ist auch möglich, mit einer RCE-Methode einen bestimmten Workflow statisch zu koppeln. Dies kann zum Beispiel zu Demonstrationszwecken sinnvoll sein.

Da der Extension-Browser alle gestarteten Methoden anzeigt, sind dort auch die im Workflow gestarteten Methoden zu sehen. Weitere Eingriffe als Starten und Stoppen sind jedoch derzeit nicht möglich. Anhalten eines Workflows könnte über eine spezielle Workflowkomponente realisiert werden, die immer ausgibt, was sie als Eingabe erhält und es über eine Java-API erlaubt, Das Weiterleiten der Eingaben zu unterbrechen bzw. wiederaufzunehmen. Der Autor müsste eine solche Komponente allerdings an einer Stelle im Workflow einbauen, an der sie sinnvoll ist. Außerdem würde es bei einem solchen Mechanismus unter umständen lange dauern, bis die Unterbrechung tatsächlich stattfindet, wenn im Eingangspuffer einer Komponente bereits mehrere Werte aufgelaufen sind. Diese würde die bereits erhaltenen Werte noch verarbeiten und die Ergebnisse ausliefern, welche dann von den folgenden Komponenten ebenfalls noch verarbeitet würden. Eine Pause-Funktion mit besserer Reaktionszeit würde erreicht, indem jede Komponente eine Pausefunktion erhalten würde, die zum Beispiel das Lesen von Werten auf Kanälen blockieren würde, auch wenn dort Werte bereitstehen. Komponenten, die an einem Wert lange rechnen, hätten allerdings auch bei dieser Variante schlechte Reaktionszeiten. Die Pause-Funktionalität in jeder Komponente von deren Autor selbst implementieren zu lassen, ist nicht empfehlenswert, da dies fehleranfällig ist.

# Kapitel 7

# Zusammenfassung

Im Rahmen dieser Master-Thesis wurden verschiedene Workflowsysteme untersucht und vorgestellt. Es wurde ein Workflowsystem, das sich in die gegebene Integrationsplattform RCE einfügt, konzipiert und implementiert. Für das Workflowsystem wurde eine graphische Benutzerschnittstelle auf Basis des STP-Modelers entwickelt. Die Integration der Benutzerschnittstelle in RCE konnte aufgrund von durch RCE noch nicht erfüllten Voraussetzungen nicht abgeschlossen werden.

Kernpunkte des Konzepts sind:

- Modellierung von Datenfluss
- Keine explizite Modellierung von Kontrollfluss
- Direkte Unterstützung von Schleifen
- Unterstützung von weiteren Kontrollflusselementen durch spezielle Komponenten
- Kommunikation zwischen Komponenten über Kanäle
- Graphische Repräsentation und Modellierung durch eingeschränkte BPMN-Diagramme

Welche der Anforderungen aus Kapitel 2 erreicht wurden und welche nicht ist in Tabelle 7.1 dargestellt.

| **Anforderung** | **Erreicht durch. . . /Gescheitert wegen. . .** | **Erreicht Ja/Nein** |
|---|---|---|
| Kommunikation mit anderen Anwendungen (Komponenten) muss für Anwendungsentwickler einfach zu realisieren sein. | Kanäle bieten eine einfache und uniforme Möglichkeit mit anderen Komponenten Daten auszutauschen. Die API und Semantik ist hierbei der Socket-Programmierung ähnlich, allerdings werden direkt Java-Objekt-Instanzen ausgetauscht. Die Kommunikation ist also für die Komponente nahezu transparent. |  |

| | | |
|---|---|---|
| Die Wiederverwendbarkeit von Komponenten muss möglichst hoch sein. | Workflowkomponenten sind normale Eclipse-Plugins (mit einigen Zusatzinformationen), die an einem bestimmten Extension-Point angebunden werden und lediglich Eingabewerte auf Ausgabewerte abbilden müssen. Eclipse-Plugins gelten als gut wiederverwertbar. Es gibt außerdem keine Einschränkungen dazu, dass die Funktionalität der Komponente nicht über andere Schnittstellen als die Kanäle nutzbar gemacht wird. Komponenten können also sogar außerhalb der Workflows wiederverwendet werden. |  |
| Es muss möglich sein, die einzelnen Komponenten in einem Workflow auf verschiedene Maschinen verteilt auszuführen. | Wird durch die RCE-Architektur und die Nutzung der RCE-Standardmechanismen zum Starten, Stoppen und Auffinden von Komponenten gewährleistet. |  |
| Es muss möglich sein, die Implementationsdetails einer Komponente vor den Partnern, die diese Komponente nutzen, zu verstecken. | Zur Nutzung einer Komponente in einem Workflow muss lediglich ihr Name und die in der *plugin.xml* abgelegten Information über verfügbare Start- und Endpunkte verfügbar sein. Auch diese müssen nicht lokal verfügbar sein, sondern können wegen der verteilten Architektur von RCE auf jeder beliebigen RCE-Instanz liegen. |  |
| Workflowkomponenten müssen wie normale RCE-Methoden verwendet werden können. | Workflowkomponenten sind normale RCE-Methoden über die weitere Zusatzinformationen vorliegen. |  |
| Das Erstellen von Workflows aus vorhandenen Komponenten muss über eine graphische Oberfläche möglichst einfach realisierbar sein. | Eine GUI wurde zwar entworfen, konnte allerdings aufgrund fehlender RCE-Bestandteile noch nicht vollständig in RCE integriert werden, so dass diese Anforderung nicht erfüllt wurde. Die nötigen Vorarbeiten sind aber so weit abgeschlossen, dass die fehlende Integration abgeschlossen werden kann, sobald die Voraussetzungen dafür seitens RCE erfüllt werden. |  |

| | | |
|---|---|---|
| Die graphische Repräsentationen des Workflows soll es erlauben, einfach auf dessen Funktion zu schließen. | Die graphische Repräsentation beschränkt sich ausschließlich auf Datenfluss. Datenzentrierte Workflows lassen sich dadurch einfach erschließen. Abbildung von kontrollflusslastigen Algorithmen direkt in dem Workflowsystem kann dadurch allerdings schwierig und unübersichtlich werden. |  |
| Das Erstellen und Verändern eines Workflows muss unabhängig von dessen Ausführung sein (Komponenten können miteinander verbunden werden, ohne dass sie tatsächlich gestartet werden). | Die GUI zum erstellen und bearbeiten der Workflows nutzt lediglich Informationen aus der *plugin.xml*-Datei der Workflowkomponente. Daher ist es nicht einmal nötig, die Klassen die die Workflowkomponenten bilden, zu laden, um einen Workflow zu erstellen bzw. zu bearbeiten. |  |
| Es muss möglich sein, Workflows zu speichern und mehrfach zu starten. | Abspeichern von Workflows als XML-Dateien wird durch bereits von STP-BPMN-Modeler unterstützt. |  |
| Workflows müssen aus dem RCE-System heraus bearbeitet werden können. | Aufgrund der fehlenden Integration der GUI in RCE bisher nicht erreicht. |  |
| Schleifen und Verzweigungen müssen realisierbar sein. | Schleifen werden direkt unterstützt. Verzweigungen können durch spezielle Workflowkomponenten realisiert werden. |  |
| Kontrollfluss und Datenfluss sollen voneinander getrennt sein. Das heißt, es soll nicht nötig sein sowohl Datenfluss als auch Kontrollfluss zu modellieren. | In dem entwickelten System wird ausschließlich der Datenfluss modelliert. |  |

Tabelle 7.1: Erreichung der Anforderungen.

Das entwickelte Workflowsystem erfüllt die gegebenen Anforderungen, wie Unterstützung von Schleifen direkt. Die geforderte Unterstützung von Verzweigungen kann durch spezielle Workflowkomponenten erreicht werden. Die hohe Wiederverwendbarkeit von Workflowkomponenten wird dadurch erreicht, dass Workflowkomponenten normale Eclipse-Plugins mit einigen Zusatzinformationen sind, die an einem bestimmten Extension-Point angebunden werden und lediglich Eingabewerte auf Ausgabewerte abbilden müssen.

Wichtige Fragestellungen, wie das Scheduling und Monitoring von Workflows konnten aufgrund ihrer Komplexität und den durch RCE zur Verfügung gestellten Mittel nicht behandelt werden. Hier bietet sich viel Raum für weitere Arbeiten.

Aufgrund des Aufbaus auf Eclipse und die Nutzung der Eclipse-Architektur konnte die Entwicklung stark

von den Vorteilen des komponentenorientierten Designs profitieren.

Im Rahmen des SESIS-Projektes werden bereits Workflowkomponenten entwickelt, um das vorgestellte Workflowsystem industriell zu nutzen.

# Anhang A

# Codelistings

Dieser Anhang enthält eine Reihe von exemplarischen Codelistings. Als Programmiersprache wird Python verwendet, obwohl die vorgestellten Algorithmen und Programmteile in Java implementiert sind. Dies liegt darin begründet, dass Python aufgrund seiner einfachen Syntax gut lesbar und somit als Pseudocode gut geeignet ist. Tatsächlich sind alle Komponenten immer Eclipse-Plugins und müssen daher in Java geschrieben werden.

Zusätzlich zu den Pythonlistings enthält dieser Anhang auch noch XML-Listings. Bei diesen handelt es sich um tatsächlich verwendbare Beispiele bzw. Auszüge aus verwendete Schemadefinitionen.

Listing A.1: *Component* sei eine Basisklasse, die den Zugriff auf die Start- und Endpunkte einer Komponente über den Channelservice transparent über einen Zugriff auf *self.startpunktname.write* bzw. *self.endpunktname.read* erlaubt und die dafür sorgt, dass die abgeleiteten Klassen ihre Ablauflogik nur noch in einer 'run'-Methode definieren müssen, so ist dies der Pseudocode für den in 5.2(a) dargestellten Workflow. Dieses Listing soll die Schritte *Definition*, *Instantiierung* und *Kopplung* von Komponenten sowie das Starten des Workflows illustrieren. Selbst bei diesem stark abstrahierten Programmierbeispiel fällt auf, dass ein Workflowsystem dem Nutzer hier viel Arbeit sparen kann. Im Idealfall muss ein Workflowautor keine oder nur wenige eigene Komponenten implementieren und kann viele vorhandene Komponenten wiederverwenden. Dies macht den Definitionsblock deutlich einfacher (ähnlich wie zum Beispiel Funktionsbibliotheken in Programmiersprachen). Instantiierung der Komponenten fällt für den Nutzer komplett weg und wird vom System übernommen. Die mitunter aufwändige und unübersichtliche Aufgabe der Kopplung wird durch eine graphische Repräsentation deutlich vereinfacht. Die eigentliche Kopplung übernimmt das Workflowsystem. Das Starten eines Workflows wird ebenfalls einfacher, da lediglich ein Button in einer GUI angeklickt werden muss. Aus einem relativ komplexen Programm wird also ein einfacher Graph. Hierbei ist zu beachten, dass sich die Fibonaccizahlen natürlich in Python auch deutlich einfacher berechnen ließen und die hier gezeigten Komponenten teilweise deutlich simpler sind als das, was in Workflowsystemen ansonsten eine typische Komponente wäre.

```
1  ######################################
2  #    Definition der Komponenten     #
3  ######################################
4  class Boot(Component):
5      def __init__(self, first_value=0)
6          self.first_value=first_value
7      def run(self):
8          self.output.write(self.first_value)
9          while not self.input.is_closed():
10             self.output.write(self.input.read())
11         self.output.close()
12
13 class Add(Component):
14     def run(self):
15         while not self.a.is_closed() and not self.b.is_closed():
16             self.c.write(self.a.read() + self.b.read())
17         self.c.close()
18
19 class Tee(Component):
20     def run(self):
21         while not self.input.is_closed():
22             input_value = self.input.read()
23             self.a.write(input_value)
24             self.b.write(input_value)
25         self.a.close()
26         self.b.close()
27
28 class Cycle(Component):
29     def run(self):
```

```
30          last_value = None
31          while not self.in.is_closed():
32              if last_value is None:
33                  last_value = self.in.read()
34              else:
35                  tmp = self.in.read()
36                  self.a.write(tmp)
37                  self.b.write(last_value)
38                  last_value = tmp
39          self.a.close()
40          self.b.close()
41
42  class PrintResults(Component):
43      def run(self)
44          while not self.input.is_closed():
45              print self.input.read()
46
47  ######################################
48  #  Instantiieren der Komponenten  #
49  ######################################
50  boot_0 = Boot(0)
51  boot_1 = Boot(1)
52  add = Add()
53  cycle = Cycle()
54  tee = Tee()
55  printer = PrintResults()
56
57  ######################################
58  #     Kopplung der Komponenten     #
59  ######################################
60  boot_0.output.connect_to(tee.input)
61  boot_1.output.connect_to(boot_0.input)
62  tee.a.connect_to(cycle.in)
63  tee.b.connect_to(printer.input)
64  cycle.a.connect_to(add.a)
65  cycle.b.connect_to(add.b)
66
67
68  ######################################
69  #       Starten des Workflows      #
70  ######################################
71  boot_0.run()
72  boot_1.run()
73  add.run()
74  cycle.run()
75  tee.run()
76  printer.run()
```

Listing A.2: Die im beispielhaft verwendeten Euklidischen Algorithmus (Abbildung 5.3(a)) verwendete *Iterator*-Komponente ist ein Beispiel für einen Mechanismus zur Simulation von Verzweigungen in datenflussbasierten Workflows. Um einen Eindruck dafür zu geben, wie einfach es ist, derartige Komponenten, die Kontrollflusselemente simulieren können, zu entwickeln sind, zeigt dieses Listing wie die *Iterator* implementiert werden würde. Python ist wie zu Beginn dieses Anhangs erwähnt lediglich als Pseudocode gedacht.

```
1  class Iterator(Component):
2      """
3      Definiert seien:
4          Endpunkt: a, b
5          Startpunkte: next, result
6
7          Ausgaben: next[n] = b[n] falls b[n] != 0
8                    result[n] = a[n] falls b[n] == 0
9      """
10     def run(self):
11         while not self.a.is_closed() and not self.b.is_closed():
12             result_value = self.a.read()
13             next_value = self.b.read()
14             if next_value != 0:
15                 self.next.write(next_value)
16             else:
17                 self.result.write(result_value)
18         self.next.close()
19         self.result.close()
```

Listing A.3: Als weitere Illustration, wie einfach Kontrollflusselemente simuliert werden können (vergleiche Listing A.2). Hier eine allgemeine *Filter*-Komponente.

```
1  class Filter(Component):
2      """
3      Definiert seien:
4          Endpunkt: input
5          Startpunkte: true, false
6          Annotation: Condition
7
8          Ausgaben: true = alle Elemente von input die 'Condition' erfuellen
9                    false = alle Elemente von input die 'Condition' nicht erfuellen
10     """
11     def run(self):
12         while not self.input.is_closed():
13             value = self.input.read()
14             if Condition(value):
15                 self.true.write(value)
16             else:
17                 self.false.write(value)
18         self.true.close()
19         self.false.close()
```

Listing A.4: Dieses Listing zeigt den Algorithmus nach dem der im Rahmen dieser Thesis entwickelte Workflowcontroller arbeitet. Im Wesentlichen werden die in Listing A.1 dargestellten Schritte aus eine übergebenen Beschreibung abgeleitet und die von RCE zur Verfügung gestellten Mechanismen zum Starten von Komponenten genutzt.

```
1  class Controller(object):
2      def __init__(self, worfkflow_description):
3          self.workflow_description = workflow_description
4          self.components = []
5
6      def start(self):
7          for key, component in self.workflow_description.components.items():
8              self.components.append(RCE.getComponent(component))
9
10         for key, connection in self.workflow_description.connections.items():
11             RCE.connect_start_and_endpoint(connection.start, connection.end)
12
13         for component in self.components:
14             component.start()
15
16     def stop(self):
17         for component in self.components:
18             component.stop()
```

Listing A.5: Dieses Listing beinhaltet die Schema-Definition für die *plugin.xml*-Dateien einer RCE-Methode. Dieses Schema beschreibt, welche Informationen in einer *plugin.xml*-Datei abgelegt werden müssen bzw. können. Einige im Rahmen dieser Thesis irrelevante Details wurden entfernt, um das Lesen zu erleichtern. Besondere Bedeutung haben in diesem Zusammenhang nur die Elemente mit dem Namen *channels*, *channelEndPoint* und *channelStartPoint*. Die Definition besagt, dass RCE-Methoden in ihrer *plugin.xml*-Datei ein XML-Element *channels* anlegen können, welches wiederum *channelStartPoint*- und/oder *channelEndPoint*-Elemente enthalten kann. Für *channelStartPoint*- und/oder *channelEndPoint*-Elemente muss eine *id* (der Name des Start- bzw. Endpunktes) und ein *type* (der von diesem Start- bzw. Endpunkt akzeptierte Java-Typ) angegeben werden.

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <schema targetNamespace="de.rcenvironment.rce.sdk">
3
4      <element name="extension">
5          <complexType>
6              <!-- Details Entfernt -->
7          </complexType>
8      </element>
9
10     <element name="platformExtension">
11         <complexType>
12             <sequence>
13                 <element ref="channels" minOccurs="0" maxOccurs="1" />
14             </sequence>
15             <!-- Details Entfernt -->
16         </complexType>
17     </element>
18
19     <element name="channels">
20         <complexType>
21             <sequence>
22                 <element ref="channelEndPoint" minOccurs="0" maxOccurs="unbounded" />
23                 <element ref="channelStartPoint" minOccurs="0" maxOccurs="unbounded" />
24             </sequence>
25         </complexType>
26     </element>
27
28     <element name="channelEndPoint">
29         <complexType>
30             <attribute name="id" type="string" use="required" />
31             <attribute name="type" type="string" use="required" />
32             <attribute name="policy" type="string" />
33         </complexType>
34     </element>
35
36     <element name="channelStartPoint">
37         <complexType>
38             <attribute name="id" type="string" use="required" />
```

```
39          <attribute name="type" type="string" use="required" />
40          <attribute name="policy" type="string" />
41        </complexType>
42    </element>
43 </schema>
```

Listing A.6: Dieses Listing zeigt wie eine *plugin.xml*-Datei für eine einfache Komponente im entwickelten Workflowsystem aussehen kann. Die *plugin.xml* bezieht sich auf die in Abschnitt 5.5 vorgestellte *Add*-Komponente zum addieren zweier Zahlen.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?eclipse version="3.2"?>
3  <plugin>
4     <extension
5           point="de.rcenvironment.rce.sdk.platformExtension">
6        <platformExtension
7              class="de.dlr.examples.Add"
8              id="de.dlr.examples.Add"
9              name="Add"
10             vendor="DLR e.V."
11             version="1.0.0">
12
13             <channels>
14                <channelEndPoint id="a" type="java.lang.Integer" />
15                <channelEndPoint id="b" type="java.lang.Integer" />
16                <channelStartPoint id="c" type="java.lang.Integer" />
17             </channels>
18       </platformExtension>
19    </extension>
20 </plugin>
```

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Literaturverzeichnis

[van der Aalst u. a. 2003] Aalst, Wil M. P. van der ; Aldred, L. ; Dumas, M. ; Hofstede, Arthur H. M. ter: Design and implementation of the YAWL system / Queensland University of Technology, Brisbane. URL http://www.yawlfoundation.org/documents/yawl_system.pdf, 2003 (FIT-TR-2003-07). – Forschungsbericht. [Online; accessed 2-Jun-2008]

[van der Aalst u. a. 2004] Aalst, Wil M. P. van der ; Aldred, L. ; Dumas, M. ; Hofstede, Arthur H. M. ter: Design and implementation of the YAWL system. In: *Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04)*. Riga, Latvia : Springer Verlag, Juni, 2004. – URL http://eprints.qut.edu.au/archive/00000379/01/aalst_yawls.pdf. – [Online; accessed 2-Jun-2008]

[van der Aalst und ter Hofstede 2003] Aalst, Wil M. P. van der ; Hofstede, Arthur H. M. ter: YAWL: Yet Another Workflow Language (Revised version) / Queensland University of Technology, Brisbane. URL http://www.yawlfoundation.org/documents/yawlrevtech.pdf, 2003 (FIT-TR-2003-04). – Forschungsbericht. [Online; accessed 2-Jun-2008]

[van der Aalst und ter Hofstede 2005] Aalst, Wil M. P. van der ; Hofstede, Arthur H. M. ter: YAWL: yet another workflow language. In: *Information Systems* 30 (2005), Nr. 4, S. 245–275. – URL http://eprints.qut.edu.au/archive/00010244/01/10244.pdf. – [Online; accessed 2-Jun-2008]

[Altintas u. a. 2004] Altintas, Ilkay ; Birnbaum, Adam ; Baldridge, Kim ; Sudholt, Wibke ; Miller, Mark A. ; Amoreira, Céline ; Potier, Yohann ; Ludäscher, Bertram: A Framework for the Design and Reuse of Grid Workflows. In: *SAG*, URL http://www.sdsc.edu/%7Eludaesch/Paper/sag04-kepler.pdf, 2004, S. 120–133

[Alves 2005] Alves, Alexandre: BPEL4WS 1.1 To WS-BPEL 2.0 - An SOA Migration Path. In: *SOA World Magazine* 706 (2005). – URL http://webservices.sys-con.com/read/155617.htm. – [Online; accessed 29-Mai-2008]

[Anjomshoaa u. a. 2005] Anjomshoaa, Ali ; Brisard, Fred ; Drescher, Michel ; Fellows, Donal ; Ly, An ; McGough, Stephen ; Pulsipher, Darren ; Savva, Andreas (Hrsg.): *Job Submission Description Language (JSDL) Specification, Version 1.0*. http://www.gridforum.org/documents/GFD.56.pdf. 2005. – [Online; accessed 29-Mai-2008]

[Apache-Software-Foundation 2008] Apache-Software-Foundation: *Apache Felix Homepage*. http://felix.apache.org/site/index.html. 2008. – [Online; accessed 3-Jun-2008]

[AWI-Bremerhaven 2008] AWI-Bremerhaven: *C3-Grid Homepage*. http://www.c3grid.de/. 2008. – [Online; accessed 29-Mai-2008]

[Booth u. a. 2004] Booth, David ; Haas, Hugo ; McCabe, Francis ; Newcomer, Eric ; Champion, Michael ; Ferris, Chris ; Orchard, David: *Web Services Architecture*. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. 2004. – [Online; accessed 29-Mai-2008]

[Bowers und Ludäscher 2005] Bowers, Shawn ; Ludäscher, Bertram: Actor-Oriented Design of Scientific Workflows. In: Delcambre, Lois M. L. (Hrsg.) ; Kop, Christian (Hrsg.) ; Mayr, Heinrich C. (Hrsg.) ; Mylopoulos, John (Hrsg.) ; Pastor, Oscar (Hrsg.): *ER* Bd. 3716, Springer, 2005, S. 369–384. – URL http://daks.ucdavis.edu/~sbowers/bowers_SWF_er05.pdf. – [online; accessed 16-Jun-2008]. – ISBN 3-540-29389-2

[Champion u. a. 2002–2004] Champion, Mike ; Hollander, Dave ; Haas, Hugo ; Booth, David: *Web Services Architecture Working Group*. http://www.w3.org/2002/ws/arch/. 2002-2004. – [Online; accessed 29-Mai-2008]

[ClimatePrediction.net 2008] ClimatePrediction.net: *ClimatePrediction.net Homepage*. http://www.climateprediction.net. 2008. – [Online; accessed 28-April-2008]

[Codehaus-Foundation 2008] Codehaus-Foundation: *Groovy Homepage*. http://groovy.codehaus.org/. 2008. – [Online; accessed 4-Jun-2008]

[Docutils 2008] Docutils: *reStructuredText*. http://docutils.sourceforge.net/rst.html. 2008. – [Online; accessed 29-Mai-2008]

[Duan u. a. 2005] Duan, Rubing ; Prodan, Radu ; Fahringer, Thomas: DEE: A Distributed Fault Tolerant Workflow Enactment Engine for Grid Computing. In: *HPCC*, 2005, S. 704–716

[Eclipse-Foundation 2008a] Eclipse-Foundation: *Eclipse GEF Homepage*. http://www.eclipse.org/gef/. 2008. – [Online; accessed 19-Jun-2008]

[Eclipse-Foundation 2008b] Eclipse-Foundation: *Eclipse Homepage*. http://www.eclipse.org/. 2008. – [Online; accessed 16-Jun-2008]

[Eclipse-Foundation 2008c] Eclipse-Foundation: *Equinox Homepage*. http://www.eclipse.org/equinox/. 2008. – [Online; accessed 3-Jun-2008]

[ETH-Zürich 2008] ETH-Zürich: *Concierege Homepage*. http://concierge.sourceforge.net/. 2008. – [Online; accessed 3-Jun-2008]

[Fahringer u. a. 2004] Fahringer, Thomas ; Pllana, Sabri ; Villazon, Alex: AGWL: Abstract Grid Workflow Language. In: *Proceedings of the International Conference on Computational Science, Programming Paradigms for Grids and Metacomputing Systems*. Krakow : Springer Verlag, 2004. – URL http://www.dps.uibk.ac.at/projects/agwl/agwlpubs/AGWL-PPGaMS2004.pdf

[Fahringer u. a. 2007] Fahringer, Thomas ; Prodan, Radu ; Duan, Rubing ; Hofer, Jürgen ; Nadeem, Farrukh ; Nerieri, Francesco ; Podlipnig, Stefan ; Qin, Jun ; Siddiqui, Mumtaz ; Truong, Hongh-Linh ; Villazon, Alex ; Wieczorek, Marek: ASKALON: A Development and Grid Computing Environment for Scientific Workflows. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 450–471

[Fahringer u. a. 2005] Fahringer, Thomas ; Qin, Jun ; Hainzer, Stefan: Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In: *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*. Cardiff, UK : IEEE Computer Society Press, May 9-12, 2005. – URL `http://www.dps.uibk.ac.at/projects/agwl/agwlpubs/junqin-ccgrid2005.pdf`

[Gannon 2007] Gannon, Dennis: Component Architectures and Services: From Application Construction to Scientific Workflows. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 174–189

[Gannon u. a. 2007] Gannon, Dennis ; Plale, Beth ; Marru, Suresh ; Kandaswamy, Gopi ; Simmhan, Yogesh ; Shirasuna, Satoshi: Adapting BPEL to Scientific Workflows. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 126–142

[Globus-Alliance 2008] Globus-Alliance: *Globus Homepage*. `http://www.globus.org/`. 2008. – [Online; accessed 18-Jun-2008]

[Goodman 2006] Goodman, Daniel J.: Martlet: A Scientific Work-Flow Language for Abstracted Parallisation. In: *Proceedings of the UK e-Science All Hands Meeting*, National e-Science Centre, 2006. – URL `http://www.climateprediction.net/science/pubs/martlet.pdf`. – [Online; accessed 20-Feb-2008]. – ISBN 0-9553988-0-0

[Goodman 2007] Goodman, Daniel J.: Introduction and evaluation of Martlet: a scientific workflow language for abstracted parallelisation. In: *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA : ACM, 2007, S. 983–992. – URL `http://www2007.org/papers/paper479.pdf`. – [Online; accessed 02-Jun-2008]. – ISBN 978-1-59593-654-7

[Grimme und Papaspyrou 2006] Grimme, Christian ; Papaspyrou, Alexander: *Workflow Specification Language*. `http://c3-grid.de/fileadmin/c3outreach/generation-0/CCCwsl_spec.pdf`. 2006. – [Online; accessed 29-Mai-2008]

[Gruber 2008] Gruber, John: *Daring Fireball: Markdown*. `http://daringfireball.net/projects/markdown/`. 2008. – [Online; accessed 29-Mai-2008]

[Hall 2008] Hall, Richard: *Oscar Homepage*. `http://oscar.objectweb.org/`. 2008. – [Online; accessed 3-Jun-2008]

[Hesse 2002] Hesse, Wolfgang: Ontologie(n) - Aktuelles Schlagwort. In: *Informatik Spektrum* 25 (2002), Nr. 6, S. 477–480. – URL http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/57/

[Hoheisel und Alt 2007] Hoheisel, Andreas ; Alt, Martin: Petri Nets. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 190–207

[Jordan u. a. 2007] Jordan, Diane ; Evdemon, John ; Alves, Alexandre ; Arkin, Assaf ; Askary, Sid ; Barreto, Charlton ; Bloch, Ben ; Curbera, Francisco ; Ford, Mark ; Goland, Yaron ; Guízar, Alejandro ; Kartha, Neelakantan ; Liu, Canyang K. ; Khalaf, Rania ; König, Dieter ; Marin, Mike ; Mehta, Vinkesh ; Thatte, Satish ; Rijn, Danny van der ; Yendluri, Prasad ; Yiu, Alex: *Web Services Business Process Execution Language Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. 2007. – [Online; accessed 29-Mai-2008]

[Kepler-Project 2003–2008] Kepler-Project: *Kepler Homepage*. http://www.kepler-project.org/. 2003-2008. – [Online; accessed 16-Jun-2008]

[Kesselman und Foster 1998] Kesselman, Carl ; Foster, Ian: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998. – URL http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/1558604758. – ISBN 1558604758

[Knuth 1997] Knuth, Donald E.: *Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition)*. Addison-Wesley Professional, July 1997. – ISBN 0201896834

[Lafon 2002–2008] Lafon, Yves: *Web Services Activity*. http://www.w3.org/2002/ws/arch/. 2002-2008. – [Online; accessed 29-Mai-2008]

[Ludäscher u. a. 2006] Ludäscher, Bertram ; Altintas, Ilkay ; Berkley, Chad ; Higgins, Dan ; Jaeger, Efrat ; Jones, Matthew ; Lee, Edward A. ; Tao, Jing ; Zhao, Yang: Scientific workflow management and the Kepler system. In: *Concurrency and Computation: Practice and Experience* 18 (2006), Nr. 10, S. 1039–1065. – URL http://www.sdsc.edu/%7Eludaesch/Paper/kepler-swf.pdf. – [online; accessed 16-Jun-2008]

[Makeweave 2008] Makeweave: *Knopflerfish Homepage*. http://www.knopflerfish.org/. 2008. – [Online; accessed 3-Jun-2008]

[Nadeem u. a. 2007] Nadeem, Farrukh ; Prodan, Radu ; Fahringer, Thomas ; Iosup, Alexandru: Benchmarking Grid Applications for Performance and Scalability Predictions. In: *CoreGRID Workshop on Middleware*. Dresden, Germany : Springer Verlag, June 2007. – URL http://www.dps.uibk.ac.at/~farrukh/publications/coreGridDresden2007.pdf. – Also published as TR-0104

[Nadeem u. a. 2006] Nadeem, Farrukh ; Yousaf, Muhammad M. ; Prodan, Radu ; Fahringer, Thomas: Soft Benchmarks-based Application Performance Prediction using a Minimum Training Set. In:

*International Conference on e-Science and Grid Computing*. Amsterdam, The Netherlands : IEEE Computer Society Press, December 2006. – URL http://www.dps.uibk.ac.at/~farrukh/publications/nadeem-softBenchmarksBasedPerformancePrediction.pdf

[OASIS 1993–2008] OASIS: *OASIS Homepage*. http://www.oasis-open.org/. 1993-2008. – [Online; accessed 29-Mai-2008]

[OMG 1997–2008] OMG: *OMG Homepage*. http://www.omg.org/. 1997-2008. – [Online; accessed 28-April-2008]

[OMG 2006] OMG: *Business Process Modeling Notation Specification*. http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf. 2006. – [Online; accessed 29-Mai-2008]

[OMG 2007] OMG: *Business Process Model and Notation (BPMN) 2.0 Request For Proposal*. http://www.bpmn.org/Documents/BPMN%202-0%20RFP%2007-06-05.pdf. 2007. – [Online; accessed 29-Mai-2008]

[OSGi-Alliance 2008] OSGi-Alliance: *OSGi Homepage*. http://www.osgi.org/. 2008. – [Online; accessed 3-Jun-2008]

[Qin u. a. 2007] Qin, Jun ; Fahringer, Thomas ; Pllana, Sabri: UML Based Grid Workflow Modeling under ASKALON. In: *Austrian-Hungarian Workshop on Distributed and Parallel Systems*. Innsbruck, Austria : Springer Verlag, September 2007. – URL http://www.dps.uibk.ac.at/~jerry/publications/AGWLTeuta-DAPSYS2006.pdf

[Reekie 1995] Reekie, Hideki J.: *Realtime Signal Processing: Dataflow, Visual, and Functional Programming*, University of Technology Sydney; School of Electrical Engineering, Abschlussarbeit zur Erreichung des Akademischen Grades Doctor of Philosophy, 1995. – URL http://ptolemy.eecs.berkeley.edu/~johnr/papers/thesis.html. – [online; accessed 2-Jan-2008]

[Russel u. a. 2007] Russel, Nick ; Hofstede, Arthur H. M. ter ; Aalst, Wil M. P. van der: newYAWL: Specifying a Workflow Reference Language using Coloured Petri Nets. In: Jensen, Kurt (Hrsg.): *Proceedings of Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*. Aarhus, Denmark : Department Of Computer Science University Of Aarhus, Oktober, 2007, S. 107–126. – URL http://www.yawlfoundation.org/documents/newYAWL-cpn.pdf. – [Online; accessed 2-Jun-2008]

[Shields 2007] Shields, Matthew: Control-Versus Data-Driven Workflows. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 167–173

[Siddiqui und Fahringer 2005] Siddiqui, Mumtaz ; Fahringer, Thomas: GridARM: Askalon's Grid Resource Management System. In: *EGC*, 2005, S. 122–131

[Slominski 2007] Slominski, Aleksander: Adapting BPEL to Scientific Workflows. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 190–207

[Taylor u. a. 2007] Taylor, Ian ; Shields, Matthew ; Wang, Ian ; Harrison, Andrew: The Triana Workflow Environment: Architecture and Applications. In: Taylor, Ian (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Secaucus, NJ, USA : Springer, New York, 2007, S. 320–339

[White 2005] White, Stephen A.: *Using BPMN to Model a BPEL Process*. `http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf`. 2005. – [Online; accessed 29-Mai-2008]

[White u. a. 2004] White, Stephen A. ; Agrawal, Ashish ; Anthony, Michael ; Arkin, Assaf ; Ball, Steve ; Bartel, Rob ; Carlsen, Steinar ; Corda, Ugo ; Fletcher, Tony ; Forgey, Steven ; Giraud, Jean-Luc ; Harmon, Paul ; Heredia, Damion ; Keeling, George ; James, Brian ; Lonjon, Antoine ; Marin, Mike ; Mason, Lee ; Miers, Derek ; Moffat, Alex ; Norin, Roberta ; Owen, Martin ; Raj, Jog ; Smith, Bob ; Sturm, Manfred ; Suryanarayanan, Balasubramanian ; Thompson, Roy ; Wuethrich, Paul Vincentand P.: *Business Process Modeling Notation (BPMN)*. `http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf`. 2004. – [Online; accessed 29-Mai-2008]

[Wieczorek u. a. 2005] Wieczorek, Marek ; Prodan, Radu ; Fahringer, Thomas: Scheduling of scientific workflows in the ASKALON grid environment. In: *SIGMOD Rec.* 34 (2005), Nr. 3, S. 56–62. – URL `http://www.sigmod.org/sigmod/record/issues/0509/p56-special-sw-section-9.pdf`. – ISSN 0163-5808

[Workflow-Patterns-Initiative 2007] Workflow-Patterns-Initiative: *Workflow Patterns Homepage*. `http://is.tm.tue.nl/research/patterns/`. 2007. – [Online; accessed 28-April-2008]

[YAWL 2008] YAWL: *YAWL Homepage*. `http://www.yawl-system.com/`. 2008. – [Online; accessed 28-April-2008]